# Universität Leipzig

Fakultät für Mathematik und Informatik

Institut für Informatik


## Simulation von Token Ring Netzwerken

Diplomarbeit


vorgelegt von: Michael Mader

geb. am 28.08.1968 in Rostock-Südstadt

angefertigt bei: Prof. Dr. Ing. W. G. Spruth, Dr. K. Hänßgen

Lehrstuhl für Technische Informatik

Universität Leipzig

# Inhaltsverzeichnis

# Einleitung

Ziel der vorliegenden Arbeit ist es, ein möglichst aussagekräftiges Modell zur Simulation eines Token-Ring-Netzwerkes zu schaffen, auf dessen Basis Untersuchungen über spezifische Eigenschaften der Datenübertragung über mehrere Ringe/Segmente hinweg mittels Bridging/Switching möglich sind, um dadurch mehr über die Qualität realer bzw. geplanter Netzwerkinstallationen bezüglich der für die jeweilige Installation verwendeten Baugruppen und/oder auf dem Markt befindlicher Produkte aussagen zu können.

Eine solche Simulation kann, in gewissen Grenzen, dazu beitragen, bestehendende Konzepte für ein geplantes Netzwerk schon im Vorhinein kostengünstig hinsichtlich ihres Gebrauchswertes zu überprüfen und sie gegebenenfalls den Erfordernissen und Bedürfnissen optimal anzupassen.

Bei der Planung und Installation von Netzwerken spielt der Kostenfaktor (zeitlicher, finanzieller und personeller Aufwand) eine nicht zu unterschätzende Rolle. Eine Überprüfung der einzelnen Konzeptionen für ein eigenes Netzwerk an realen Geräten kommt somit für die meisten Firmen bzw. Institutionen aus mehreren Gründen nicht in Betracht. Fehlkalkulationen hinsichtlich der Leistungsfähigkeit und Verwendbarkeit von bestimmten Komponenten bzw. Mängel in der Konzeption und Installation werden oft erst relativ spät (im laufenden Betrieb) deutlich. Das kann zu hohen finanziellen Verlusten führen und ist somit aus heutiger Sicht inakzeptabel.

Ein Simulationsmodell kann hierbei sehr zeit- und kostensparend die wesentlichen Eigenschaften und Reaktionen des Netzwerkes im Betrieb verdeutlichen. Allerdings ist eine zusätzliche Evaluierung herstellerspezifischer Gerätetypen damit nicht notwendigerweise ausgeschlossen, da bevorzugt modernste Technik eingesetzt wird und diese aufgrund von immer kürzeren Entwicklungszeiten Charakteristika zeigen können, welche auch anhand einer Simulation nicht ohne weiteres nachvollziehbar sind. Gründe hierfür können unter anderem Fehler in der Hardware bzw. Software (z.B.: Micro-Code) sein.

Aber eine Simulation bietet (je nach Ansatz und Ausführung) nicht nur die Möglichkeit, das Gesamtverhalten eines Netzwerkes sichtbar zu machen, sondern es ermöglicht auch

Einblicke in systeminterne Prozesse, was zur effizienteren Gestaltung und Nutzung des geplanten realen Systems (hinsichtlich möglicher Netzwerk- bzw. Datenstrukturen und somit Hardware- und Softwareeinsatz) beitragen kann.

Durch ein besseres Verständnis der systeminternen Prozesse ist es möglich, ein für die eigenen Belange maßgeschneidertes System aufzubauen bzw. ein bestehendes System veränderten Bedingungen anzupassen und so vorhandene Resourcen möglichst effektiv einzusetzen.

Aus vorgenannten Gründen ist es vorteilhaft, anhand einer Simulation die wichtigsten Eigenschaften des Netzwerkes offenzulegen. Je nach Betrachtungsweise und Zielstellung können verschiedene Ansätze für ein solches Simulationsmodell gewählt werden.

In den folgenden Abschnitten, sollen Einsatzmöglichkeiten, Vor- und Nachteile der einzelnen Ansätze dargestellt werden. Ausgehend von den theoretischen Grundlagen in Abschnitt 1 werden verschiedene Möglichkeiten der Beschreibung von Netzwerken bzw. einzelnen Komponenten eines Netzwerkes aufgezeigt. Hierzu wurden vor allem die Publikationen von H.M. Deitel [Dei90] und N. Drakos [Vas96] herangezogen. Im Abschnitt 2 wird erläutert, weshalb gerade das vorliegende Modell für die Realisierung der Simulation gewählt wurde und wie es hinsichtich seiner Charakteristika in die Vielzahl der Simulationen einzuordnen ist. Die in diesem Abschnitt verwendeten Klassifikationsansätze beruhen im allgemeinen auf den in P. Luksch‘s Arbeit über die „Parallelisierung ereignisgetriebener Simulationsverfahren…“ [Luk93] erörterten Klassifikationen. Im weiteren Verlauf wird dieses Modell eingehend beschrieben. Später werden verschiedene Beispielmessungen an mittels Simulator nachgebildeten einfachen Netzwerken diskutiert. Die Ergebnisse aus den Simulationen können Aufschluß über Eigenschaften und Gesetzmäßigkeiten unterschiedlicher Übertragungsprotokolle und –verfahren geben. Die gewonnenen Einblicke in Mechanismen, die in dem simulierten Netzwerk wirken und die daraus resultierenden Erkenntnisse, können später unter Umständen im realen System Anwendung finden. Dies schlägt sich nieder in der Wahl der geeigneten Hardware, Software, Netzwerktopologie, Übertragungsprotokolle und den günstigsten Parametern für diese.

Erkenntnisse aus den im Rahmen dieser Arbeit durchgeführten Simulationen werden in den Abschnitten 3 und 4 diskutiert.

# 1 Theoretische Grundlagen

In diesem Abschnitt wird näher auf die Hintergründe der Simulation von Netzwerken und sequenziellen Prozessen und deren Realisierungsmöglichkeiten mittels verschiedener Modellansätze eingegangen werden. Hierzu gehören mathematische ebenso wie empirische Herangehensweisen. Jeder Ansatz bietet gegenüber den anderen Vor- und Nachteile, die im folgenden dargestellt und ausgewertet werden sollen.

## 1.1 Modellansätze

Bei der Schaffung eines Modells zur Simulation von sequenziellen Prozessen, und im speziellen von Netzwerken, existieren verschiedene Möglichkeiten eines Ansatzes. Vor allem auch hinsichtlich der zur Beschreibung verwendeten Kriterien [Luk93] können diese stark variieren. Ausgehend von der Modellskalierung und Granularität der Simulation unterscheidet man zwischen der *mikroskopischen*, der *mesoskopischen* und der *makroskopischen* Betrachtungsweise. Die Sichtweise und der gewählte Modellansatz entscheiden über die Aussagefähigkeit und Einsatzmöglichkeit der Simulation, deshalb ist es wichtig, daß im Vorhinein festgelegt ist, welchen Zweck man mit der Simulation verfolgt, welche Art von Resultaten man erwartet und wie präzise die Simulation das reale System wiedergeben soll.

Die *makroskopische* Betrachtungsweise beruht auf der Sicht auf das System als Ganzes, vergleichbar mit einer „Black Box“. Es werden nur die äußeren Charakteristika des Systems betrachtet. Technische Einzelheiten finden hier im allgemeinen kaum Berücksichtigung. Ereignisse in diesem System werden als rein zufällig angesehen und beschrieben. Solch ein System kann mathematisch beschrieben werden, das bedeutet, alle Komponenten des Systems (ihre Funktionsweise) werden durch mathematische Formeln wiedergegeben. Wie schon erwähnt, führt das aber dazu, daß das System auch nur recht grob beschrieben werden kann. Es ist in diesem Fall nicht oder nur unzureichend möglich die Interna der Komponenten (Geräte) bzw. Einzelheiten bezüglich der Eigenschaften des Datenflusses und einzelner Individuen (Datenrahmen) innerhalb des Datenstromes darzustellen. Mit dieser

Betrachtungsweise ist es, bezogen auf das vorliegende Problem, nur möglich allgemeine Aussagen über das System hinsichtlich der Beziehung zwischen Datenaufkommen, Systemauslastung und daraus resultierendem Datendurchsatz zu machen, es ist also eher dazu geeignet die Belastbarkeit eines System grob hinsichtlich seiner Struktur darzustellen.

Die *mikroscopische* Sichtweise greift auf eine möglichst detailgetreue Beschreibung eines Systems, seiner Komponeten und deren Funktionen zurück. Hier werden nicht nur mathematische Formeln eingesetzt, um Komponenten des Systems darzustellen, als vielmehr die problemrelevanten Mechanismen möglichst genau nachgebildet. Das ermöglicht eine ausgesprochen realitätstreue Darstellung diskreter Zustände in dem Modell und damit die Realisierung und Betrachtung einzelner „Individuen“ (Datenrahmen) innerhalb eines Verarbeitungsstroms. Somit sind Aussagen über dynamische Eigenschaften des modellierten Systems bzw. einzelner Komponenten innerhalb des Systems möglich. Da hier diskrete Zustände eines Systems dargestellt werden können, ist dieser Ansatz unter anderem geeignet für die Simulation von Warteschlangensystemen (eng.: *queueing systems*), Kommunikationsnetzen und Hardware-Systemen/Komponenten. Durch diese Detailtreue ist es ebenfalls möglich, das System mittels seiner Parameter optimal einzustellen und so den eigenen Bedürfnissen anzupassen.

Die *mesoscopische* Betrachtungsweise stellt einen Mittelweg zwischen den beiden obengenannten Herangehensweisen dar. Zum einen wird in diesem Fall eine mathematische Beschreibung des Systems gewählt, aber auf der anderen Seite beinhaltet dieser Ansatz auch schon das Einstreuen von einzelnen Individuen in die Simulation, was dazu führt, daß auch Aussagen über das dynamische Verhalten des Systems und seiner Komponenten möglich sind. Eine Möglichkeit, ein entsprechendes Modell zu verwirklichen, ist durch die Schaffung von Objekten (Individuen), welche zum Zeitpunkt ihrer Generierung durch das System einen Zeitstempel erhalten [Bha95], auf den dann verschiedene Funktionen angewendet werden, während das Individuum das System durchläuft, erreicht. Hat ein solches Objekt (auch Actor genannt) das System komplett durchlaufen, wird sein nun veränderter Zeitstempel ausgewertet.

## 1.2 Mathematische Hintergründe

Im folgenden werden Möglichkeiten aufgezeigt, wie Netzwerke simuliert werden können, indem sie makroscopisch bzw. mesoscopisch beschrieben werden. Das System wird durch das Heranziehen von Zufallsverteilungen zur Beschreibung von Ankunfts- bzw. Verarbeitungsprozessen dargestellt.

### 1.2.1 Warteschlangentheorie

Warteschlangen (engl.: *queue*) treten immer dann auf, wenn in einem System eine sequentielle Verarbeitung stattfindet, bei der die Verarbeitungsgeschwindigkeit zumindest partiell unter der allgemeinen Verarbeitungsgeschwindigkeit des Systems liegt oder bei der mehrere Verarbeitungsströme zusammengeführt werden, sodaß es zu Stauungen im Verarbeitungsfluß kommen kann.
In der Warteschlangentheorie (engl.: *queueing theory*) werden die zu verarbeitenden Objekte, in Anlehnung an die allgemeine Vorstellung einer Warteschlange, als *Kunden* und die Verarbeitung als *Bedienung* oder auch *Service* bezeichnet [Dei90].
Warteschlangen können in ihrer Länge *beschränkt* (engl.: *bounded*) oder *unbeschränkt* (engl.: *unbounded*) sein. Die Anzahl der in der Schlange wartenden Kunden kann im Falle einer unbeschränkten Warteschlange unendlich groß werden bzw. im Falle der beschränkten Warteschlange nur bis zu einer festen Größe ($n \geq 0$) anwachsen.
Was uns im Falle des Auftretens von solchen Warteschlangen (im folgenden auch als *Queues* bezeichnet) interessiert, ist, wie lange das System benötigt, um einen bzw. mehrere Kunden zu bedienen, also wie lange ein Kunde im System zubringt.
In der Warteschlangentheorie werden zur Darstellung der benötigten Zeit für komplexe Verarbeitungsprozesse Zufallszahlen herangezogen, die durch Wahrscheinlichkeitsverteilungen beschrieben werden können.

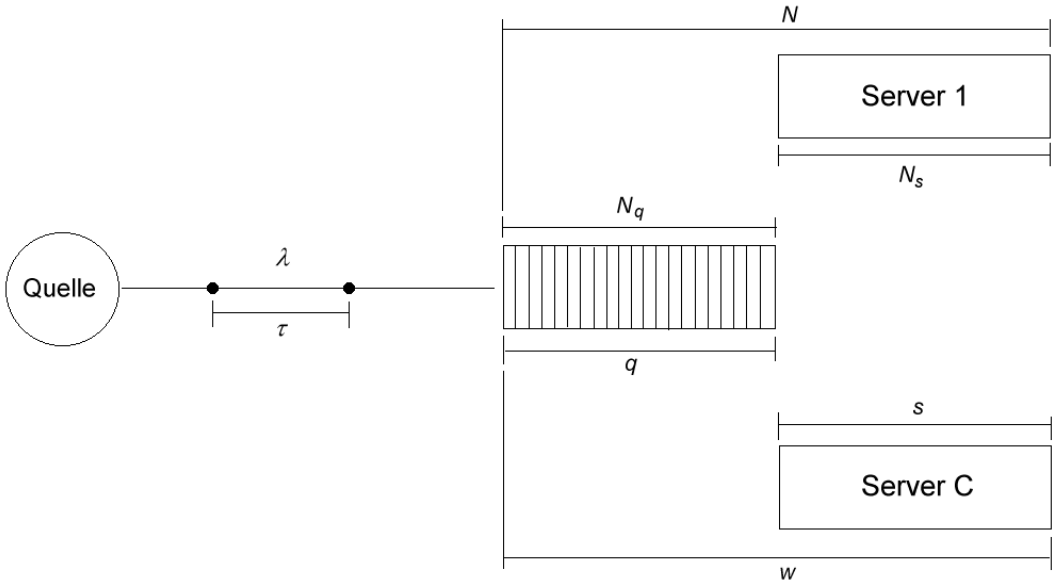


**Abbildung 1.2.1:** Einige der wichtigsten Variablen eines Warteschlangenmodells und deren Umgebungsbereiche [Dei90]: $\lambda$– durchschnittliche Ankunftrate; $\tau$ – Ankunftzeitenabstand; $N$ – Gesamtzahl der Kunden im Warteschlangensystem; $N_q$ – Anzahl der Kunden in der Warteschlange; $N_s$ – Anzahl der Kunden in der Verarbeitung; $q$ – In der Warteschlange verbrachte Zeit; $s$ – im Service verbrachte Zeit; $w$ – Gesamtzeit die ein Kunde im Warteschlangensystem verbringt.

So verbringt ein Kunde zum Beispiel eine gewisse, als zufällig angenommene Zeit $q$ in der Warteschlange bevor er in den Verarbeitungsprozeß eintritt, um dann noch eine bestimmte (ebenfalls als zufällig angenommene) Zeit $s$ in der Verarbeitung selbst zu verbringen (siehe Abbildung 1.2.1).

Wenn $w$ die Gesamtzeit darstellt, die ein Kunde im System verweilt, so gilt [Dei90]:

$$w = q + s \; . \qquad (1)$$

Nun interessiert uns nicht nur die Zeit, die ein Kunde in einem solchen System zubringt, sondern wir möchten ganze Ströme von Kunden (Datenströme) betrachten, welche im allgemeinen in einem Netzwerk auftreten. Da der Zeitpunkt des Eintreffens und die Verweilzeit eines einzelnen Kunden im System zufällig ist, können sich in einem Warteschlangensystem zu einem Zeitpunkt mehrere Kunden aufhalten. Die Gesamtzahl $N$ der Kunden im ganzen System wird bestimmt durch die Anzahl $N_q$ der Kunden, die sich derzeit in der Warteschlange befinden und die Anzahl $N_s$ der Kunden, die sich gerade in der Verarbeitung durch einen der Server des Systems befinden, bestimmt (Abbildung 1.2.1).

Der Einfachheit halber nimmt man an: (a) daß die Menge der Kunden in der Quelle (engl.: *Source*) unendlich groß ist, und (b) daß die Kunden sequentiell in das System

eintreten. Daraus folgt, daß bei hinreichend klein gewählter Länge des Zeitintervalls (im weiteren auch als Zeiteinheit bezeichnet) immer nur ein Kunde zu einer Zeit $t_k$ ($k \geq 0$) im System ankommt, wobei die diskreten Zeitpunkte $t_k$ eine aufsteigende Folge bilden (vergl.: Definition Markov-Prozeß / Markov-Ketten [Mat90]):

$$t_0 < t_1 < t_2 < \ldots < t_n . \tag{2}$$

Wir können nun eine durchschnittliche Ankunftrate festlegen bzw. ermitteln, mit welcher die Kunden das System erreichen. Diese ist herleitbar aus den einzelnen Differenzen $\tau_k$ der Ankunftzeiten jeweils aufeinanderfolgender Kunden ($t_k$ - $t_{k-1}$ , $k \geq 1$), die in das System erfolgreich eingetreten sind.

$$\tau_k = t_k - t_{k-1} , \qquad (k \geq 1) . \tag{3}$$

Alle Zufallswerte sind voneinander unabhängig und gleichverteilt. Im allgemeinen wird in der Warteschlangentheorie davon ausgegangen, daß diese Ankunftszeiten durch einen POISSON-Prozeß [Dei90, Mat90, Vas96] beschrieben werden. Ein POISSON-Prozeß ist wie folgt definiert: Wenn $P(k,t)$ die Wahrscheinlichkeit ist, daß genau $k$ Ankünfte in einem Zeitintervall der Länge $t$ auftreten

$$P(k,t) = \begin{cases} \lambda\Delta t & \text{für } k = 1 \\ 1 - \lambda\Delta t & \text{für } k = 0 , \\ 0 & \text{für } k > 1 \end{cases} \tag{4}$$

wobei $k \in \mathbb{N}_0$, $k \geq 0$ ist und die Ereignisse, die als in nichtüberlappenden Zeitintervallen $\Delta t$ definiert wurden, wechselseitig unabhängig sind. Aus dieser Definition folgt, POISSON-Prozesse stellen sequentielle Verarbeitungssysteme dar. Wird das Zeitintervall $\Delta t$ hinreichend klein gewählt, so kann $k$ nur die Werte 0 oder 1 annehmen. Auf die Warteschlangentheorie bezogen bedeutet das, daß immer nur ein Kunde in einem Zeitintervall im System eintreffen bzw. verarbeitet werden kann.

POISSON-Ankunftsprozesse sind dadurch charakterisiert, daß die Zeiten zwischen jeweils zwei erfolgreichen Ankünften von Kunden im System, im folgenden auch als Ankunftszeitenabstand $\tau$ (engl.: *interarrival times*) bezeichnet, exponentiell verteilt sind

$$P(\tau \leq t) = 1 - e^{-\lambda t}\,, \tag{5}$$

und die Wahrscheinlichkeit, daß genau *n* Kunden in jedem Zeitintervall der Länge *t* das System erreichen ist

$$\frac{e^{-\lambda t}\,(\lambda t)^n}{n!}\,, \qquad (n = 0, 1, 2, \ldots), \tag{6}$$

wobei die durchschnittliche Ankunftsrate $\lambda$ ein fester Wert ist und ausgedrückt wird in „Kunden pro Zeiteinheit“. Die Anzahl der Ankünfte pro Zeiteinheit wird als POISSON-verteilt bei durchschnittlichem $\lambda$ beschrieben [Dei90]. Den Wert für die durchschnittliche Ankunftrate erhält man aus der Beziehung

$$\lambda = \frac{1}{E(\tau)} \tag{7}$$

$E(\tau)$ ist hier die erwartete Zeit zwischen zwei erfolgreichen Ankünften von Kunden im System (*expected interarrival time*).

Wie die Ankunftszeiten werden auch die Verarbeitungszeiten als zufällig angenommen, weiterhin wird angenommen, daß auch diese exponentiell verteilt sind, somit gilt

$$W_\tau(t) = P(s \leq t) = 1 - e^{-\mu t}\,, \qquad (t \geq 0)\,. \tag{8}$$

$\mu$ ist hier die durchschnittliche Servicerate des Servers und ergibt sich aus dem Reziproken des Erwartungswertes für die Servicezeit eines Kunden

$$\mu = \frac{1}{E(s)} \tag{9}$$

Gehen wir von einer infiniten Kundenquelle (Datenquelle) aus, so behandeln wir einen unendlich langen Prozeß.

Nimmt man nun den Grenzübergang für die Verteilungsfunktion für die Verarbeitungszeiten vor, so ist die Wahrscheinlichkeit, daß genau *n* Kunden im Zeitintervall *t* verarbeitet werden

$$\frac{e^{-\mu t}(\mu t)^n}{n!}, \qquad (n = 0, 1, 2, \ldots) . \tag{11}$$

Die Auslastung $\rho$ eines Servers eines Systems, ergibt sich aus dem Verhältnis aus Datenaufkommen (Verkehrsdichte) und der Verarbeitungsfähigkeit der einzelnen Server, also durch ihre Anzahl *x* , da von einer gleichverteilten Verarbeitungsfähigkeit ausgegangen wird

$$\rho = \frac{u}{x} , \tag{12}$$

wobei *u* die Verkehrsdichte darstellt und sich aus dem Verhältnis zwischen Ankunfts- und Bedienrate der Server herleitet

$$u = \frac{\lambda}{\mu} . \tag{13}$$

In einem 1-Server-System würde das somit wie folgt aussehen:

$$\rho = \frac{\lambda}{\mu} . \tag{14}$$

Weitere Eigenschaften die in Warteschlangensystemen berücksichtigt werden müssen, sind

- die Kapazität der Warteschlangen

- die Anzahl der Server im System
- die Verarbeitungsregel der Warteschlange.

Hieraus ergibt sich ein Klassifizierungssystem das durch die Kendall-Notation beschrieben werden kann [Dei90]:

A/B/x/K/m/Z

| | |
|---|---|
| *A* | ist die Ankunftszeitverteilung (engl.: *interarrival time distribution*) |
| *B* | ist die Bedienzeitverteilung (engl.: *service time distribution*) |
| *x* | ist die Anzahl der Server im System |
| *K* | ist die Kapazität der Warteschlange des Systems |
| *m* | ist die Anzahl der Kunden |
| *Z* | ist die Abarbeitungsregel innerhalb der Warteschlange. |

Gebräuchliche Bezeichner für Verteilungsarten die für A bzw. B eingesetzt werden, sind im folgenden aufgelistet:

- *GI* für allgemein unabhängige (engl.: *general independent*) Ankunftszeit
- *G* für allgemeine (engl.: *general*) oder willkürliche (*arbitrary*) Bedienzeit (Paketlängen)
- $E_k$ für Erlang-*k* Ankunfts- bzw. Bedienzeitverteilung
- *M* für exponentielle Ankunfts- bzw. Bedienzeitverteilung (Markovsystem)
- *D* für deterministische Ankunfts- bzw. Bedienzeitverteilung
- $H_k$ für hyperexponentielle Ankunfts- bzw. Bedienzeitverteilung (*k*-ter Stufe)

Gewöhnlich wird in der Warteschlangentheorie von M/M/x (nach Kendall-Notation) Systemen ausgegangen. Die Ankunftszeitverteilung *A* und die Bedienzeitverteilung *B* sind exponentiell und bilden somit Markovsysteme *M*. Die Anzahl der Server *x* im System ist größer oder gleich 1. Da die Anzahl der Kunden als unendlich angenommen wird, wird die Anzahl der Kunden *m* in der Quelle gewöhnlich nicht explizit angegeben.
In einem M/M/x System ($x \geq 1$) geht man von exponentiell verteilten Ankunftszeiten und Verarbeitungszeiten aus. Die Wahl der Verteilungsart beruht auf der Annahme, daß die Kunden (in unserem Fall Datenrahmen) zu nicht vorhersagbaren Zeiten das

Warteschlangensystem erreichen, begründet zum einen durch eine Vielzahl von konkurrierenden Datenströmen und zum anderen durch das nicht vorhersagbare Auftreten von Daten in einem einzelnen Datenstrom.

Weiterhin wird diesbezüglich vorausgesetzt, daß die einzelnen Zufallsvariablen in einem solchen M/M/x System ($x \geq 1$) voneinander unabhängig sind. In realen Netzwerken treten jedoch auch Systeme mit zwei oder mehr hintereinandergeschalteten Warteschangen (*tandem links* [Vas96]) auf, je nach Detailtreue der Simulation und Komplexität des nachzubildenden Netzwerkes.

In einem solchen Fall, werden zum einen die Voraussetzungen hinsichtlich der Unabhängigkeit der Zufallsvariablen verletzt, und zum anderen kann das Modell nicht mehr den Anforderungen des realen Systems entsprechen.


**Abbildung 1.2.2:** System zweier aufeinanderfolgender Warteschlangen (Tandem Link Queueing System). In diesem Falle sind die Ankunfts- und Verarbeitungsverteilungen stark voneinander abhängig, was zu einer Verletzung der Prozeß- bzw. Verteilungsfunktions-definition führt. Ist $Q_1$ ein M/M/1-System, so kann $Q_2$ keins sein [Vas96].

Betrachtet man beispielsweise ein System von zwei aufeinander folgenden Warteschlangen $Q_1$ und $Q_2$ (Abb. 1.2.2), ist zu erkennen, daß wenn $Q_1$ ein M/M/1 System darstellt, so kann $Q_2$ kein M/M/1 System sein, da Ankunfts- und Servicezeiten stark voneinander abhängig sind [Vas96], was durch die geforderte Unabhängigkeit dieser Werte automatisch ausgeschlossen wird.

Aus diesem Beispiel wird deutlich: Je komplizierter ein Netzwerk strukturiert ist, also je zufälliger Datenrahmen bzw. ganze Folgen von Datenrahmen, im folgenden auch als Datenströme bezeichnet, von verschiedenen voneinander unabhängigen Quellen auftreten, desto einfacher ist es, dieses mittels mathematisch-stochastischer Methoden (z.B. Warteschlangentheorie) zu modellieren.

In komplexen Systemen mit ausgedehnter Topologie kann davon ausgegangen werden, daß die einzelnen Warteschlangen mehrere Quell- bzw. Zielwarteschlangen besitzen, falls diese sich nicht an der Peripherie des Systems befinden. Auf diese Weise werden Abhängigkeiten aufeinander folgender Warteschlangen verhindert. Hierbei wird eine weitere Eigenschaft von POISSON-Prozessen ausgenutzt.

Die Komposition oder Vereinigung mehrerer POISSON-Prozesse bildet wiederum einen POISSON-Prozeß. In Abb. 1.2.3 sind die Vereinigung (Komposition) und die sich hieraus ergebenden Relationen zwischen den einzelnen Warteschlangensystemen (Prozessen) dargestellt.

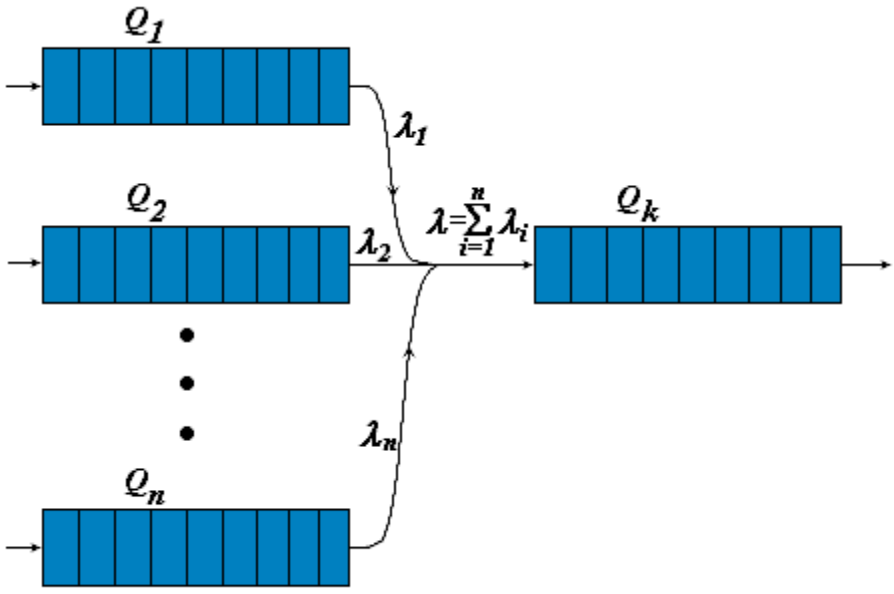


**Abbildung 1.2.3:** Komposition von POISSON-Prozessen [Dei90], wie sie in Warteschlangensystemen auftreten können. Jede Warteschlange symbolisiert einen POISSON-Prozeß.

Eine Komposition von POISSON-Prozessen kann zum Beispiel die Vereinigung von Datenströmen in Konzentratoren, Multiportbrücken oder anderen Knotenpunkten in einem Netzwerk beschreiben. Unter Knotenpunkten sind hier Orte (Geräte) zu verstehen an denen Datenströme sich kreuzen bzw. miteinander verschmelzen, sich also gegenseitig beeinflussen können. Oft bilden solche Knotenpunkte eine Kombination aus Komposition und Dekomposition von Datenströmen (Prozessen).

Die Dekomposition, also das Aufspalten von POISSON-Prozessen (Abb. 1.2.4, siehe auch [Dei90]), sieht ähnlich aus. Jeder der aus der Dekomposition resultierenden Ausgangsdatenströme (Eingangsdatenströme der Folgeprozesse) tritt mit einer bestimmten spezifischen Wahrscheinlichkeit aus dem System, in dem die Dekomposition stattfindet, aus. Diese spezifische Wahrscheinlichkeit manifestiert sich in einem Faktor $p_j$ mit dem $\lambda$ jeweils

multipliziert wird und so die einzelnen $\lambda_j$ bildet, wobei die Summe aller Wahrscheinlichkeiten $p_j$ ($0 \le p_j < 1$) den Wert 1 ergeben muß

$$\sum_{j=1}^{n} p_j = 1 , \qquad (15a)$$

$$\lambda = \sum_{j=1}^{n} p_j \lambda = \lambda \sum_{j=1}^{n} p_j . \qquad (15b)$$

Die $p_j$ müssen nicht notwendigerweise Wahrscheinlichkeiten für die Wahl des Weges zu einer der folgenden Warteschlangen sein, ebenso ist es möglich, daß das aktuelle Datenpaket (Kunde) das Netzwerk verläßt, da das Paket für den aktuellen Netzwerkknoten, der durch die Warteschlange repräsentiert wird, bestimmt ist.

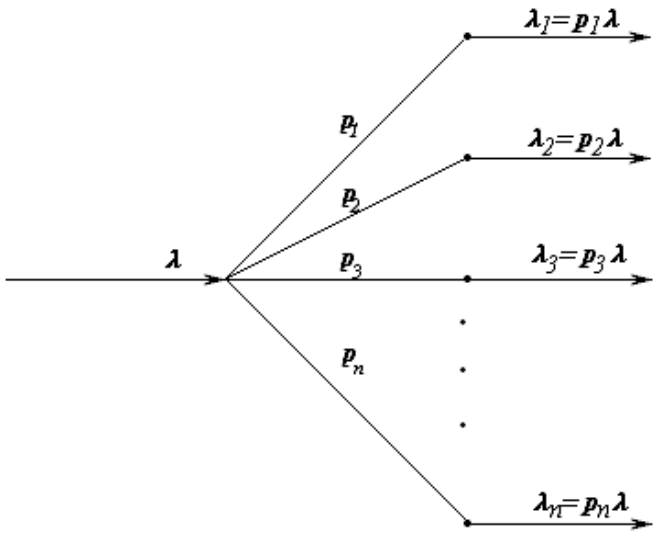


**Abbildung 1.2.4:** Dekomposition von POISSON-Prozessen in *n* unabhängige Prozesse [Dei90]. Die $p_j$ geben die jeweilige Ereignis- oder Selektionswahrscheinlichkeit für die Netzwerkkante (Pfad) $K_j$ an, wobei (15a) gelten muß.

Ebenso ist es möglich, daß keine direkte Verbindung zwischen den Warteschlangen $Q_i$ und $Q_j$ besteht, dann wäre die Wahrscheinlichkeit $p_{ij}$ der Wahl dieser Route gleich 0. Somit kann man diese Formel erweitern und ihre Gültigkeit auf das gesamte Netzwerk ausdehnen.
Wenn *K* die Anzahl aller im Netzwerk vorkommenden Warteschlangen darstellt und $p_{id}$ die Wahrscheinlichkeit ist mit der das Paket am aktuellen Knoten das Netzwerk verläßt (Knoten selbst ist das Ziel), so gilt [Dei90,Vas96]:

$$p_{\mathrm{id}} + \sum_{i=1}^{K} p_{ij} = 1 \; . \qquad (16)$$

Die Dekomposition stellt weiterhin ein Mittel dar, mit dem man Teilaspekte im Datenverkehr von sich sternenförmig oder hierarchisch (baumartig) ausbreitenden Netzen bzw. Teilnetzen simulieren kann.

Ein Netzwerk das obige Bedingungen erfüllt und durch die genannten Formeln beschrieben werden kann, wird auch als JACKSON-Netzwerk bezeichnet. Zusammengefaßt ist ein JACKSON-Netzwerk, ein System mit *K* Warteschlangen, das die folgenden 3 Bedingungen erfüllt [Vas96]:

- Die Bedienzeiten jeder der *K* Warteschlangen sind voneinander unabhängig mit Raten $\mu_1, \mu_2, ..., \mu_K$ .
- Die externen Ankünfte (falls vorhanden) an jeder Warteschlange sind unabhängige POISSON-Prozesse mit Raten $\tau_1, \tau_2, ..., \tau_K$ ($\tau_i \geq 0$ , für $i = 1, 2, ..., K$).
- Das Routing im Netzwerk ist zufällig (engl.: *random routing*).

Ein JACKSON-Netzwerk besteht aus voneinander unabhängigen M/M/1-Warteschlangensystemen. Somit gelten hier auch alle entsprechenden Gesetzmäßigkeiten, wie z.B. die Unabhängigkeiten der Paketgrößen bzw. Verarbeitungszeiten in den verschiedenen Warteschlangensystemen.

Ein solches komplexes Netzwerk aus unabhängigen Warteschlangensystemen können wir unter Verwendung von Jackson's Theorem [Vas96] zur Bestimmung der Gesamtankunftsrate $\lambda_i$ :

$$\lambda_i = \tau_k + \sum_{j=1}^{K} \lambda_j p_{ij} \; , \qquad (17)$$

mit ($i = 1, 2, ..., K$) und Little's Formel [Vas96, Dei90]

$$E(T) = \frac{E(n)}{\lambda} = \frac{\rho}{\lambda(1-\rho)} = \frac{1}{\mu - \lambda} \qquad (18)$$

wobei $E(n)$ die durchschnittliche Gesamtanzahl der Pakete im Netz bzw. Subnetz darstellt, und

$$E(n) = \sum_{i=1}^{K} E(n_i) \tag{19}$$

uns auch als ein einziges M/M/1-Warteschlangensystem vorstellen, wobei die mittlere Ende-zu-Ende-Verzögerung eines Paketes durch

$$E(T) = \frac{E(n)}{\gamma} = \frac{1}{\gamma} \sum_{i=1}^{K} E(n_i) = \frac{1}{\gamma} \sum_{i=1}^{K} \frac{\lambda_i}{\mu_i - \lambda_i} \tag{20}$$

beschrieben werden kann.

Der Nachteil hier liegt wiederum in dem hohen Maß an Zufälligkeit (Grundlage dieser Theorie), was es erschwert spezielle Netzwerkinstallationen nachzubilden. Insbesondere kleinere Netze bzw. Netzwerke mit speziellen Charakteristika bezüglich Hard- und Softwarenutzung sind hiermit nur sehr unzureichend simulierbar.

Will man spezielle Hardware hinsichtlich ihrer Einsatzfähigkeit im Netz testen, ist es oft notwendig, Einblicke in das dynamische Verhalten des Netzes zu erhalten. Auf diese Weise können Spitzenbelastungskurven erstellt und somit eventuelle Flaschenhälse aufgespührt und beseitigt werden (entweder durch Umstrukturierung des Netzes oder durch Hard- bzw. Softwareaustausch).

## 1.2.2 Birth and Death-Prozesse

Ein Warteschlangensystem kann auch beschrieben werden als ein System, das sich zu einem bestimmten Zeitpunkt $t_k$ in einem bestimmten Zustand $S_k$ aus einer vollständig abgeschlossenen Menge exklusiver diskreter Zustände $\{S_0, S_1, S_2, \ldots, S_n\}$ befindet. Das Verhalten eines solchen Systems kann durch einen MARKOV-Prozeß beschrieben werden.
Eine spezielle Form von MARKOV-Prozessen bilden die sogenannten *Birth and Death*-(Geburts- und Sterbe-) Prozesse, im folgenden auch als BD-Prozesse bezeichnet. Diese Art von Prozessen ist besonders geeignet zur Modellierung von Computersystemen. Sie sind wesentlich einfacher zu behandeln als gewöhnliche MARKOV-Prozesse. Ein fortlaufender BD-Prozeß besitzt die folgende Eigenschaft

$$\lambda_{ij}=0 , \qquad \text{falls } (j \neq i+1 \text{ und } j \neq i-1) , \tag{21}$$

wobei $\lambda_{ij}$ die Rate ist, mit der Übergänge vom Zustand $S_i$ zum Zustand $S_j$ auftreten. Also ist die Übergangsrate $\lambda_{ij}$ nicht aufeinander folgender Zustände $S_i$, $S_j$ gleich Null. Dies geht auch aus Abbildung 1.2.5 hervor.

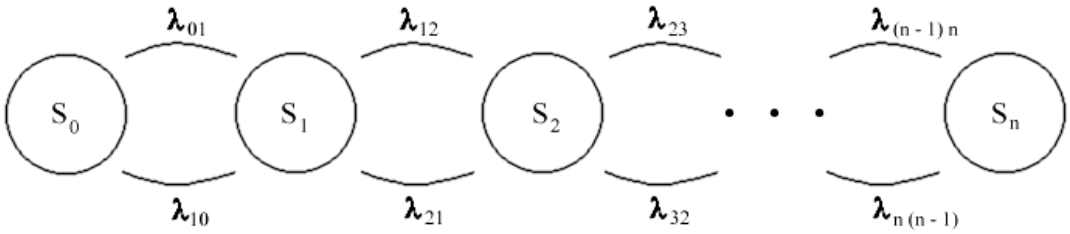


**Abbildung 1.2.5:** Schematische Darstellung eines Birth and Death- Prozesses [Dei90]. $S_i$ bezeichnet die verschiedenen Zustände; $\lambda_{i\,(i+1)}$ die durchschnittliche Geburtsrate des Zustands $S_I$ ; $\lambda_{i\,(I-1)}$ die durchschnittliche Sterberate des Zustands $S_I$ .

Für solche Prozesse benutzt man im allgemeinen die Notation

$$\lambda_{i(i+1)} = b_i = \text{durchschnittliche Geburtsrate vom Zustand } S_i \tag{22}$$

$$\lambda_{i(i-1)} = d_i = \text{durchschnittliche Sterberate des Zustands } S_i . \tag{23}$$

# 2 Modellbeschreibung und Klassifikation

In diesem Abschnitt soll ausgehend von den Gegebenheiten im realen (Token Ring-) Netzwerksystem die theoretische und technische Umsetzung von Übertragungs- bzw. Verarbeitungsmechanismen im Simulationsmodell aufgezeigt werden. Im ersten Teil werden hierbei die protokollspezifischen und im zweiten Abschnitt die gerätespezifischen Aspekte, die Eingang in das Simulationsmodell gefunden haben, diskutiert. Abschliessend wird eine Klassifizierung des erarbeiteten Modells und des dieses Modell realisierenden Simulators vorgenommen. Als Grundlage für die Klassifikation des Simulators diente vor allem die Arbeit von P. Luksch [Luk93], in welcher unter anderem allgemeingültige Kriterien für die Klassifikation ereignisgesteuerter Simulatoren gefunden und erörtert wurden.

## 2.1 Modellierung problem-relevanter Protokoll-mechanismen

Als Ausgangspunkt für die Modellierung des Übertragungsprotokolls in der Simulation wurde das NetBIOS-Protokoll gewählt, da die vorliegende Arbeit in Zusammenhang mit den Untersuchungen zur Evaluierung von TokenRing-Netzwerkkomponenten von T. Schmidt [Sch97] zu sehen ist, in welchen ebenfalls NetBIOS als Übertragungsprotokoll eingesetzt wurde. Die in [Sch97] gewonnenen Ergebnisse und Erkenntnisse dienten als Bezugspunkt für die Verifikation des vorliegenden Modells. Um entscheiden zu können, welche Eigenschaften des Protokolls Berücksichtigung im Modell finden müssen, ist es unabdingbar, Kenntnis über die Zusammenhänge innerhalb und zwischen den einzelnen Protokollschichten zu haben. Bei der Bezeichnung und Charakterisierung der einzelnen Schichten wurde vom OSI-Schichtenmodell ausgegangen (siehe Abbildung 2.1.1).

Demzufolge werden die zu versendenden Dateneinheiten in Anlehnung an die OSI-Terminologie wie folgt bezeichnet:

| | |
|---|---|
| Schicht 5 | Nachricht |
| Schicht 4 | Segment |
| Schicht 3 | Paket |
| Schicht 2 | Rahmen. |

Bei NetBIOS fallen die Schichten 3 und 4 zusammen. Eine Dateneinheit wird innerhalb dieser Schicht als Paket bezeichnet.

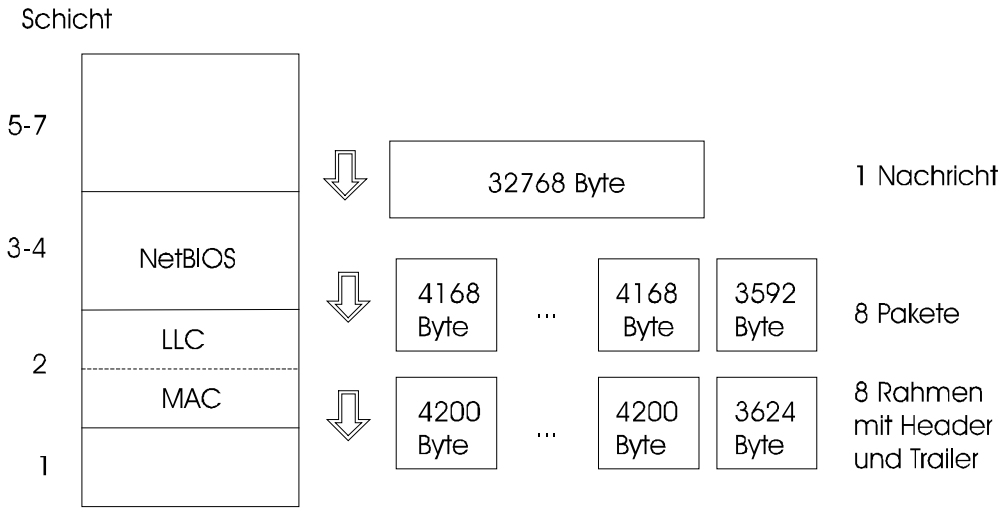


**Abbildung 2.1.1:** Schematische Darstellung - Protokollschichten und Paketgenerierung unter NetBIOS [Sch97].

Das NetBIOS-Protokoll greift, wie in Abbildung 2.1.1 zu sehen, auf die Dienste der Verbindungsschicht (Schicht 2; engl.: *Data Link Layer*) [Sch97, RFC02, Ker95, Hod21] zu. Die Verbindungsschicht wird weiter unterteilt in die *MAC* (*Medium Access Control*)-Schicht und die *LLC* (*Logical Link Control*)-Schicht. Sie hat die Aufgabe, eine *Type 2 Verbindung* zwischen den beiden Rechnern zu gewährleisten. Eine einfache *Type 2 Verbindung* ist in Abbildung 2.1.2 dargestellt. Die Kommunikation zwischen den beiden Gegenstellen auf dieser Protokollebene wird über das Versenden von Rahmen (engl.: *Frames*) realisiert. Man unterscheidet hierbei die Gruppe der unnummerierten Rahmentypen und die Gruppe der Informationsrahmentypen (nummerierte Rahmentypen). Zu den unnummerierten Rahmentypen zählen zum einen jene, die für den Kommunikationsauf- und –abbau verantwortlich zeichnen (siehe Abb. 2.1.2), und zum anderen Rahmen, die zum Test der Funktionstüchtigkeit (z.B. Loopback-Test) dienen. Zu diesen Typen zählen unter

anderem SABME-, XID-, UA-, FRMR-, TEST- und DISC-Frames (siehe Abb 2.1.2). Da diese Gruppe von Rahmentypen in dem Simulationsmodell keine Berücksichtigung findet, soll an dieser Stelle auch nicht näher auf diese Typen eingegangen werden.

Zu der Gruppe der Informationsrahmentypen (auch nummerierte Rahmentypen genannt) zählen Datenrahmen (*Data Frames*) und Datenkontroll- bzw. Steuerrahmen (engl.: *Control Frames* oder *Data Control Frames*). Die Datenrahmen kann man wiederum nach ihrer logischen Position innerhalb eines Datenpaketes unterteilen in Rahmen, die am Anfang bzw. an einer mittleren Position lokalisiert sind (engl. Bez.: *DataFirstMid - Frame*), und in Rahmen, die am Ende eines Paketes stehen (engl. Bez.: *DataOnlyLast – Frame*), zu letzteren zählen auch all jene Rahmen die ein kompletes Datenpaket in sich vereinigen.

Unter dem Begriff der Datenkontrollrahmen wird eine Vielzahl von Rahmentypen zusammengefaßt, die wichtigsten innerhalb einer laufenden Datenübertragung (von der in diesem Modell stets ausgegangen wird) sind die Antwort- oder Bestätigungsrahmen (engl.: *Acknowledgement Frames* oder kurz *Acknowledges bzw. ACK-Frames*). Diese Rahmen bestätigen dem Sender den korrekten Empfang der Datenrahmen durch den Zielrechner. Bestätigungsrahmen werden von unterschiedlichen Protokollschichten versandt. Während NetBIOS (Schicht 3 und 4) den Empfang kompletter Pakete mittels *ACK*-Rahmen bestätigt, wird auf der LLC-Ebene (Verbindungsschicht; Schicht 2) zu diesem Zweck ein *RR*-Rahmen (*receiver ready*; dtsch.: „Empfänger bereit“) gesendet [RFC02, Vas96]. Dieser Rahmen enthält unter anderem die Sequenznummer des zuletzt empfangenen Datenrahmens bzw. Datenpaketes. Deshalb werden diese Rahmen auch als *LLC-Sequence Frame* bezeichnet.

Nachdem sich die Kommunikationspartner über die Übertragungsparameter verständigt haben, beginnt der eigentliche Datentransfer, der nach folgenden Regeln arbeitet:

1. Die zu sendenden Pakete werden in Rahmen unterteilt die kleiner oder gleich der *maximalen Rahmengröße* sind (inkl. Header und Trailer).

2. Die einzelnen Rahmen werden vom Empfänger quittiert entweder per *ACK-Frame* auf NetBIOS-Ebene (falls Rahmentyp *DataOnlyLast*) oder mittels *LLC-Sequence-Frame* (falls Rahmentyp *DataFirstMid*).

3. Die empfangenen Pakete werden (wie auch aus (2.) schon hervorgeht) quittiert mit einem Bestätigungsrahmen (*ACK-Frame*).

4. Die *ACK-Frames* werden wiederum vom Sender quittiert (*LLC-Sequence Frame*)

5. Die *LLC-Sequence-Frames* enthalten entweder die Sequenznummer der einzelnen Rahmen (aus (2.)) oder die der Pakete (aus (4.)).

6. Es können so viele Datenrahmen des Typs *DataFirstMid* vom Sender übertragen werden, ohne auf ein Acknowledgement zu warten, wie als Wert für die Fenstergröße (*Window*) bei Verbindungsaufbau festgelegt wurden (siehe Abb. 2.1.2; XID-Rahmen).

Nachdem die eigentliche Datenübertragung beendet ist, beginnt der Verbindungsabbau. Hierzu gehören die Beendigung der NetBIOS-Session und die Trennung der Verbindung auf LLC-Ebene (siehe Tabelle 2.1.1).

| Sender | Empfänger | Rahmentyp | Rahmen-größe | Erklärung |
|---|---|---|---|---|
| WS2 | NetBIOS | NETB Add name query TECH52-- | 63 | Server wartet auf Anfrage. |
| WS2 | WS1 | NETB Name query TECH52__ | 61 | Anfrage vom Client. |
| WS1 | WS2 | NETB Name recognisedLSN 8C | 61 | |
| WS2 | WS1 | LLC: SSAP: F0 DSAP: F0 SABME C1 | 17 | Anfrage zum Verbindungsaufbau. |
| WS1 | WS2 | LLC: SSAP: F0 DSAP: F0 UA R 1 | 17 | Nicht nummerierte Bestätigung. |
| WS2 | WS1 | LLC: SSAP: F0 DSAP: F0 RR C 1 00 | 18 | |
| WS1 | WS2 | LLC: SSAP: F0 DSAP: F0 RR R 1 00 | 18 | |
| WS2 | WS1 | NETB Session init88_8C | 32 | Aufbau der NetBIOS-Session. |
| WS1 | WS2 | NETB Session confirm8C_88 | 32 | Session akzeptiert. |
| WS2 | WS1 | LLC: SSAP: F0 DSAP: F0 RR R 0 01 | 18 | Sender bereit. |
| WS2 | WS1 | NETB Data only last | 92 | Datenübertragung |
| WS1 | WS2 | NETB Data ack | 32 | läuft mit einer Paket- |
| WS2 | WS1 | LLC: SSAP: F0 DSAP: F0 RR R 0 02 | 18 | größe von 64 Byte. |
| | | . | | |
| | | . | | |
| | | . | | |
| WS2 | WS1 | NETB Session endNormal | 32 | Session beenden. |
| WS1 | WS2 | LLC: SSAP: F0 RR R 0 6B | 18 | |
| WS2 | WS1 | LLC: SSAP: F0 DISC C 1 | 17 | Verbindung beenden. |

**Tabelle 2.1.1:** Ablauf einer NetBIOS-Datenübertragung (Mitschnitt[Sch97]). Die grau unterlegten Zeilen zeigen den Verbindungsauf- bzw. –abbau. In diesem Fall werden 337 Byte während des Verbindungsaufbaus und 67 Byte während des Verbindungsabbaus übertragen.

Für die Simulation der Übertragung mittels NetBIOS sind Verbindungsaufbau bzw. -abbau von untergeordneter Bedeutung (bei hinreichend großer Datenmenge), da es sich hierbei zum einen um eine für jede Datenverbindung relativ gleiche Datenbelastung handelt (Abweichungen bestehen nur aus 1 – 2 Rahmen mit einer durchschnittlichen Größe von 18 Byte) , unabhängig von der Rahmengröße, und zum anderen diese Belastung relativ gering ist. Ausgehend von einem typischen Verbindungsaufbau, wie er in Abb 2.1.2 dargestellt ist und einer angenommenen zu übertragenden Datenmenge von 500 Kbyte, beträgt der prozentuale Anteil von Verbindungsaufbau und –abbau am Gesamtdatenaufkommen während einer Datenübertragung weniger als 0,07 % .

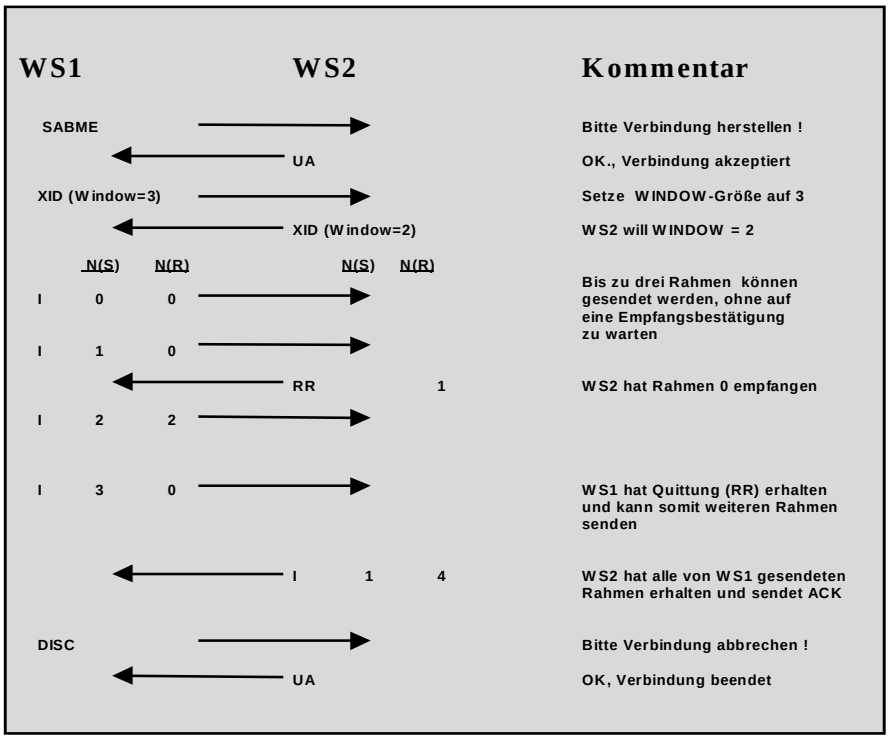


**Abbildung 2.1.2**: Ablauf einer herkömmlichen Typ-2 Verbindung mit Verbindungsaufbau [Vas96], Datenübertragung und Verbindungsabbau. Rahmenbezeichner haben folgende Bedeutung: SABME - Set Asynchronous Balanced Mode Extended, Anfrage zum Etablieren einer Verbindung; UA - Unnumbered Acknowledgment, nicht nummerierte Antwort auf SABME, falls Empfänger Verbindung akzeptiert; XID - Exchange Identification, Kommando und Antwort, wird benutzt von Typ 1 und Typ 2; RR – Receiver Ready, zeigt an, daß Empfänger bereit für Empfang des nächsten Rahmens ist; DISC – Disconnect, Kommando zum Schliessen einer Verbindung,wird mit UA oder DM beantwortet; DM – Disconnect Mode, Antwort auf SABME, falls Empfänger Anfrage ablehnt.

Wie aus Abb. 2.1.2 ersichtlich, ist der zeitliche Ablauf der Kommunikation abhängig von der Wahl der *Rahmenfenster* (engl.: *Window*) - *Größe*. Der Wert für das Rahmenfenster entscheidet darüber, wieviele aufeinanderfolgende Datenrahmen (innerhalb eines zu übermittelnden Datenpaketes) vom Sender an den Empfänger geschickt werden dürfen, ohne auf eine Bestätigung durch den Empfänger zu warten (vergl. [Tan92]). Durch

geeignete Wahl des Wertes für diesen Parameter können die Rechenkapazität der beiden Kommunikationspartner und das Übertragungspotential der Netzwerkverbindung besser ausgenutzt werden, da der Sender zur Generierung aufeinander folgender Datenrahmen nicht in jedem Fall auf eine vorherige Antwort des Empfängers in Form eines Kontrollrahmens mit der Sequenznummer des zuvor gesendeten Datenrahmens warten muß. Das führt dazu, daß die Kommunikation „flüssiger“ abläuft und das Netz besser ausgelastet wird.

Wie der Fenstermechanismus so hat auch das Übertragsprotokoll, das zur Datenübermittlung zwischen den Kommunikationspartnern eingesetzt wird, einen entscheidenden Einfluß auf das Übertragungsverhalten. Da dies wiederum Einfluß auf den Datendurchsatz (im Realsystem) hat, wurde diesen beiden Aspekten im Simulationsmodell auch besondere Aufmerksamkeit geschenkt.

Sender und Empfänger wurden so konzipiert, daß sie auf die Art der ankommenden Rahmen reagieren, das heißt, je nachdem, ob der Sender ein Rahmen vom Typ *DataFirstMid* oder vom Typ *DataOnlyLast* an den Empfänger schickt, reagiert dieser, indem er ein *LLC-Sequence-Frame* (bei *DataFirstMid*) bzw. ein *ACK-Frame* (bei *DataOnlyLast*) als Bestätigung an den Sender zurückschickt.

Der Daten sendende Rechner reagiert auf die vom Empfänger ankommenden Rahmen, indem er je nach Rahmentyp einen weiteren Datenrahmen oder ein *LLC-Sequence-Frame* als Bestätigung des *ACK-Frames* sendet. Dabei wurde aber kein besonderer Wert auf den Inhalt der *LLC-Sequence-Frames* gelegt, ausgenommen die Einträge für Quelle und Ziel, da im Modell keine Vertauschungen in der Reihenfolge der Datenrahmen innerhalb der Kommunikation zusammengehöriger Partner auftreten können. Relevant sind in diesem Fall nur die Rahmengrößen und der allgemeine Kommunikationsablauf (wie schon erläutert).

Der Fenstermechanismus wird im Modell durch de- bzw. inkrementieren einer Variable (GOT_ACK) realisiert. Ist der Inhalt dieser Variablen Null, so muß auf eine Rahmenbestätigung durch den Empfänger gewartet werden, ist der Wert größer, darf der Sender weiter Datenrahmen senden.

Ebenfalls als wichtig für eine möglichst realitätsnahe Simulation erwies sich die Berücksichtigung der weiter unten angesiedelten Protokollschichten (Eigenschaften des Übertragungsmediums, der Flußsteuerung und -kontrolle).

Die Funktionen dieser Schichten wurden stark vereinfacht in das Modell übernommen. So wurden zum Beispiel die Mechanismen zur Behandlung von Netzwerkfehlern außer Acht

gelassen, da in dem Modell von einem korrekt implementierten, funktionierenden System ausgegangen wurde, bei dem im Normalfall keine Broadcast-Stürme (das vermehrte Auftreten von Rundsendungen) oder ähnliche Phänomene auftreten. Somit sind die Einflüsse durch Kontrollmeldungen und -abfragen relativ gering und, was noch wichtiger ist, die Netzbelastung ist verhältnismäßig konstant. Somit reduzieren sich die darzustellenden Funktionen auf:

1. die Gewährleistung einer sequenziellen Datenübertragung zwischen jeweils zwei Kommunikationspartnern, die anhand einer eindeutigen Adresse identifiziert werden können, welche im Kopf jedes Rahmens, der gesendet wird, enthalten ist.

und

2. die Steuerung des allgemeinen Datenflusses auf dem Ring mittels kreisendem Token, welches durch die Boolsche Variable *TOKENFREE* realisiert wurde, die in Abhängigkeit davon, ob der Ring gerade von einem Übertragungsprozeß genutzt wird oder nicht, auf *TRUE* bzw. *FALSE* gesetzt wird.

Im vorliegenden Modell wird eine Datenübertragung zwischen den einzelnen Stationen im Store & Forward-Modus durch die Funktion *MoveFrameThroughWire( )* auf dem Ring und innerhalb einer Station mittels der Funktion *MoveFrameTo( )* simuliert. Im Cut-Through-Übertragungsmodus kommt die Funktion *CutThroughMove( )* zum Einsatz.
Der Unterschied der Funktionen besteht in der Art, wie die Daten übertragen werden. Wie schon aus den Bezeichnern der einzelnen Funktionen erkennbar, simuliert *MoveFrameThroughWire( )* die byteweise Datenübertragung zwischen den einzelnen Netzwerkkomponenten, also die Übertragung über den „Wire“ (Draht). In Verbindung mit der Funktion *MoveFrameTo( )*, die das Kopieren der Daten innerhalb eines Gerätes vom Eingangspuffer an den Ausgangspuffer und die Freigabe des Eingangspuffers für neue Daten darstellt, wird so eine Store & Forward-Übertragung [Sch97] simuliert.
Die Funktion *CutThroughMove( )* dient im Gegensatz dazu der Realisierung eines Systems, das im Cut-Through-Modus [Gaw95, Cor97, Sch97, Mess95] arbeitet. Daten die im Eingangspuffer entgegengenommen werden, können sofort weitergeleitet werden, sobald eine gewisse Anzahl von Bytes empfangen wurde (Einlesen der Kopfdaten des zu übermittelnden Rahmens) bzw. eine gewisse Zeit vergangen ist (Zieladressenermittlung).

## 2.2 Modellierung hardware-technischer Mechanismen

Wie aus Messungen an realen Systemen [Sch97] ersichtlich wurde, schlagen sich die Systemcharakteristika der einzelnen Geräte (Rechner, Bridge/Switch), wie z.B. Speicherbestückung und Systemarchitektur, in den Meßergebnissen nieder, da hierdurch das Antwortzeitverhalten der einzelnen Kommunikationspartner bestimmt wird. Deshalb muß dieser Faktor auch in der Simulation Berücksichtigung finden. In diesem Zusammenhang müssen zum Beispiel auch Teile der Speicherverwaltung innerhalb der Bridges/Switches (Shared Memory) und ihre möglichen Auswirkungen auf den allgemeinen Datenfluß im System betrachtet werden. Hierbei ist es wichtig zu wissen, ob eventuell Blockierungen der anderen Datenströme (zwischen anderen Kommunikationspartnern) auftreten können.
Aus den Messungen an realen Systemen [Sch97] geht auch hervor, daß je leistungsfähiger die Kommunikationspartner sind (Paketgenerierung), desto höher ist die prozentuale Auslastung der einzelnen Ringe des Systems (durch besseres Antwortzeitverhalten bzw. höheren Datenausstoß), auf denen die Kommunikation abläuft.

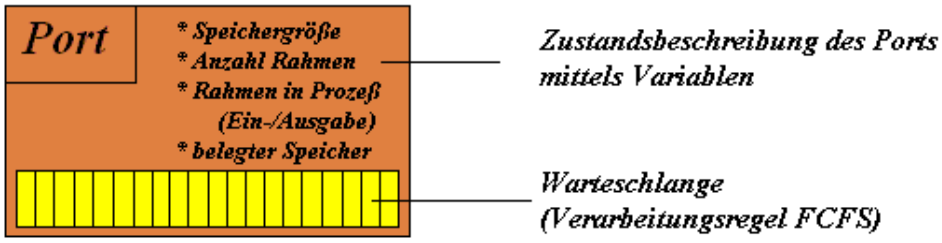


**Abbildung 2.2.1:** Schematische Darstellung eines Sende- bzw. Empfangsports innerhalb eines Gerätes im Simulationsmodell.

Die einzelnen Netzwerkknoten (Geräte) werden dargestellt durch eine Kombination aus das Gerät beschreibenden Zustandsvariablen, einer Menge von FIFO (*first in first out*)-Puffern, die jeweils eine Warteschlange mit FCFS (*first come first served*)-Verarbeitungsregel simulieren und einer Funktion die, je nach Gerätetyp und Übertragungsverfahren, die ankommenden Daten entweder verarbeitet bzw. weiterleitet (dem nächsten Puffer/Warteschlange übergibt). Die Warteschlangen in Verbindung mit bestimmten Zustandsvariablen bilden Teilsysteme, die in Anlehnung an reale Systeme als Ports bezeichnet werden (Abbildungen 2.2.1 und 2.2.2). Ein Port ist eine Struktur mit einer Menge von Zustandsvariablen und einer Warteschlange die als Listenstruktur [Lan89,

Els88] realisiert ist. Zu den einen Port beschreibenden Zustandsvariablen gehören unter anderem Angaben über die maximale und die derzeitig genutzte Speichergröße, die Anzahl der sich derzeit im System befindlichen Datenrahmen, die Angabe, ob gerade ein Datenrahmen verarbeitet wird oder nicht, die Anzahl der bisher verlorengegangenen Datenrahmen und Angaben über die topologische Einbettung des einzelnen Ports in die Gesamtstruktur des simulierten Netzes, wie z.B. Ringzugehörigkeit und spezifischer Bezeichner (gleichzusetzen mit Adresse).

Handelt es sich bei der darzustellenden Komponente um eine Bridge oder einen Switch, so sind die einzelnen Ports durch eine Funktion miteinander verbunden, die einen Datentransport mit einer dem Gerät und Transfermodus entsprechenden Verzögerung simuliert. Die dazu verwendeten Funktionen wurden in Abschnitt 2.1 beschrieben.

So bilden sich Paare aus Eingangs- und Ausgangsports. Abbildung 2.2.2 stellt diese Relation noch einmal schematisch dar.

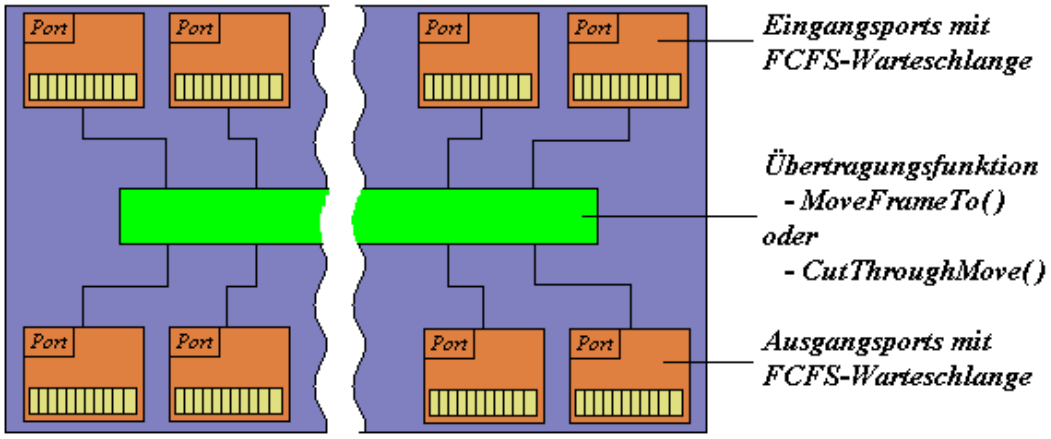


**Abbildung 2.2.2**: Schematische Darstellung eines Gerätes wie z.B. Bridge oder Switch

Da in modernen Netzkopplungselementen das Prinzip des Shared Memory (gemeinsam genutzter Speicher) eingesetzt wird, werden im Simulationsmodell Informationen bezüglich der aktuellen Speichernutzung der einzelnen Ports an das übergeordnete System weitergegeben und so zu einer Gesamtauslastung des Speichers aufsummiert, aber auch eine Einzelauswertung der Speicherauslastung der einzelnen Ports ist auf diese Weise möglich.

Die Abarbeitung von Microcode innerhalb der Geräte durch das Systemmanagement wird im Modell durch eine zeitliche Verzögerung (engl.: *Delay*) in der Weiterleitung der Daten (vom Eingangspuffer zum Ausgangspuffer) nachgebildet.

Abweichungen in der Verzögerungszeit der zu übertragenden Daten, die in solchen Geräten zum Beispiel durch die Adreßauflösung oder andere Managementprozesse auftreten, werden durch Zufallswerte beschrieben.

Die Werte für die Speichergröße, die Standardverzögerung durch die Verarbeitung und die Abweichung sind variabel, so daß mittels geeigneter Werte bestimmte Gerätetypen bzw. Produktreihen nachgebildet werden können.

Die vorgenannte topologische Einbettung der Ports und Geräte wird im vorherein mittels einer Initialisierungsdatei festgelegt. Die Topologie bestimmt den Programmablauf der Simulation durch die Festlegung der Reihenfolge der Warteschlangen im Verarbei-

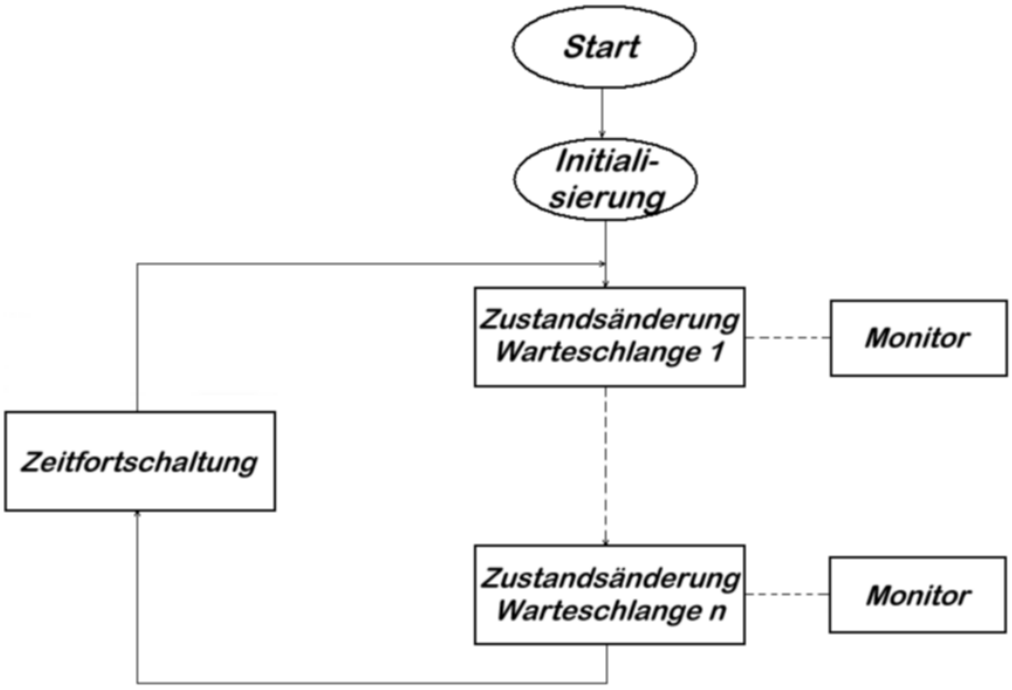


**Abbildung 2.2.3:** Schematische Darstellung der Ablaufsteuerung des Simulationsmodells.

tungsschema (siehe Abbildung 2.2.3). Ausgehend von der den Sender des Meßrechnerpaares repräsentierenden Warteschlange (darauf folgend die Sender der einzelnen Lastrechnerpaare der selben Ebene) wird das gesamte System abgearbeitet. Somit ist die allgemeine Abarbeitungsrichtung vom jeweiligen Sender zum entsprechenden Empfänger. Die Abarbeitungsrichtung entspricht auf diese Weise der generellen Richtung des Datenflusses.

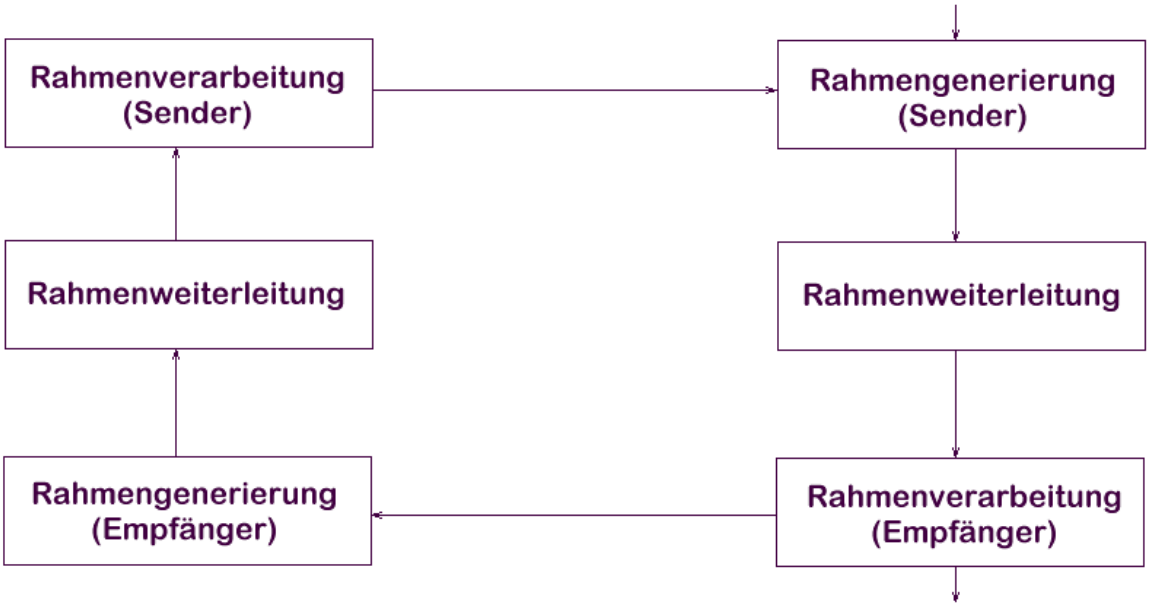


**Abbildung 2.2.4:** Schematische Darstellung einer einfachen Kommunikation aus Sicht der Rahmenverarbeitung.

Die Abbildung 2.2.4 zeigt die Modellabläufe aus Sicht der Rahmenverarbeitung. Ein Rahmen wird vom Sender erzeugt, der Rahmen wird durch die verschiedenen Funktionen, die den zurückzulegenden Pfad des Rahmens repräsentieren, an den Empfänger übergeben und dort ausgewertet. Jenachdem, was für ein Rahmen empfangen wurde (Datenrahmen oder Kontroll-/Steuerrahmen) wird ein entsprechender Rahmen vom Empfänger für den Sender generiert.

## 2.3 Klassifikation von Modell und Simulator

Um den gewählten Modellentwurf gut verständlich und möglichst konkret in das Feld der (real/theoretisch) bestehenden Simulationen einordnen zu können, wurden verschiedene Klassifikationsschemata herangezogen. Sie wählen unterschiedliche Ansätze zur Beschreibung eines Modells. Da solche Klassifikationen meist mit Blick auf eine ganz bestimmte Lösungsmenge (Modellen) vorgenommen wurden, erfassen sie im allgemeinen nicht die Gesamtheit aller Modelle.

So kann das Simulationsmodell bespielsweise hinsichtlich seiner programmier-technischen Realisierung charakterisiert und klassifiziert werden, unabhängig davon welcher Art das der Simulation zugrundeliegende Problem ist und auf welche Weise das Problem behandelt wird. Eine solche Klassifikation wird unter anderem in [Luk93] beschrieben.

Das vorliegende Modell realisiert nach Luksch einen CM/EI/ZS – Entwurf [Luk93]. Das bedeutet, die Simulation wird sequentiell im Compiled-Mode (CM) abgearbeitet: Bei jedem Schleifendurchlauf werden alle Teilobjekte des Modells ausgewertet, unabhängig davon, ob eine veränderte Eingangsbelegung für das jeweilige Objekt vorliegt oder nicht. Die Reihenfolge der Auswertung der einzelnen Teilobjekte wird während der Systeminitialisierung festgelegt und ist unveränderlich für einen kompletten Simulationsdurchlauf. Die Wahl des Compiled-Mode macht sich besonders bei der Simulation von einem hohen, möglichst symmetrisch auf dem System verteilten Datenaufkommen positiv bemerkbar, da in diesem Fall die Wahrscheinlichkeit veränderter Eingangsbelegungen für jedes simulierte Teilobjekt sehr hoch ist.

Dem gegenüber würde bei Verwendung einer Ereignissteuerung anstatt des Compiled-Mode der Verwaltungsaufwand mit zunehmendem zu simulierenden Datenaufkommen stark ansteigen, da für jedes zu simulierende Objekt die entsprechende Ereignisliste durchsucht und gegebenenfalls modifiziert werden muß. Dies führt zur Verschiebung des Verhältnisses zwischen Simulation und Verwaltung zugunsten der Verwaltung, was sich negativ auf die Leistung des gesamten Systems auswirkt.

Ist das Datenaufkommen aber sehr niedrig und unsymmetrisch über das zu simulierende Netzwerk verteilt und/oder das zu simulierende System sehr komplex, so kann die Wahl einer ereignisbasierten Ablaufsteuerung wiederum von Vorteil sein, da nur Elemente, mit veränderter Eingangsbelegung, sogenannte aktive Elemente, in dem jeweiligen Simulationszyklus betrachtet werden.

Da das Ziel eines Simulationsdurchlaufes in den meisten Fällen aber das Auffinden von Belastungsgrenzen für ein gegebenes System ist und darüber hinaus oft auch nur Teile eines Netzwerkes betrachtet (simuliert) werden, kann davon ausgegangen werden, daß alle bzw. der überwiegende Teil der Elemente des Simulationsmodelles zu jedem Zeitpunkt aktiv sind. Hiervon ausgenommen sind jeweils der Initial- und Terminalzustand der Simulation. Somit werden die Vorteile der Verwendung einer Ereignissteuerung gegenüber dem Compiled-Mode zumindest im Falle eines sequenziellen Simulatormodelles weitestgehend egalisiert.

Die Zeitfortschaltung im vorliegenden System wird zentral (ZS) nach jedem Programmzyklus mittels Einheitsinkrement (EI) vorgenommen (Abb. 2.2.3). Somit entfällt das sonst notwendige Synchronisieren der Zeit innerhalb der einzelnen Systemebenen bzw. zwischen den verschiedenen simulierten Objekten (Geräten/Mechanismen). Neben einer

einfacheren Verwaltung des Systems besteht ein weiterer Vorteil dieser Art der Zeitfortschaltung in der so möglich werdenden Betrachtung und Auswertung der verschiedenen Systemparameter und –zustände zu jedem beliebigen Simulationszeitpunkt. Als Nachteil dieser Variante ist die geringere Simulationsgeschwindigkeit zu nennen. Trotz allem ist das Verhältnis von Real- zu Simulationszeit (1:50 bei Simulation einer Datenübertragung über ein 2-Ring-System ohne weitere Netzlast) im allgemeinen noch recht günstig im Hinblick auf den gewählten Granularitätsgrad (Genauigkeit der Nachbildung) des Modells. Die Simulationszeit nimmt mit jedem weiteren simulierten Datenstrom circa um den Faktor 2 (abhängig von der Datenlast und Anzahl der zu durchlaufenden Netzknoten) zu. Das bedeutet die für einen Simulationsdurchlauf benötigte Zeit nimmt mit jedem auftretenden Ereignis (zu verarbeitende Daten) zu. Hierauf wird im Abschnitt 3 noch näher eingegangen.

# 3 Beispielsimulationen

In diesem Kapitel werden verschiedene durchgeführte Beispielsimulationen vorgestellt und deren Ergebnisse diskutiert. Abschnitt 3.1 erläutert noch einmal die Motivation für die Auswahl der Art und Durchführung der Experimente. In den Abschnitten 3.2 – 3.4 werden die einzelnen Experimente eingehend vorgestellt und deren Ergebnisse besprochen. Es wird das Systemverhalten bezüglich des Datendurchsatzes und der Speicherbelastung der Ein- und Ausgangsports der Netzkopplungsgeräte bei Simulation einer Datenübertragung in den Modi „Store and Forward“ (S&F) und „Cut-Through“ (CT) untersucht.

## 3.1 Grundlage der Simulationsmessungen

Als Ausgangspunkt für die Realisierung und spätere Kalibrierung des vorliegenden Simulators dienten die Meßergebnisse und Erfahrungen aus der Arbeit von T. Schmidt [Sch97], da die Ergebnisse aus diesen Tests und Messungen wegen der einfachen Versuchsanordnung nachvollziehbar und reproduzierbar sind. Die Tests wurden in einem abgeschlossenen TokenRing Netzwerk mit minimaler Ringlänge und unter ausschließlicher Verwendung von NetBIOS als Übertragungsprotokoll vorgenommen. Durch die Abgeschlossenheit des Systems wurde einem eventuellen Auftreten konkurrierender Fremdkommunikation im Versuchsnetz entgegengewirkt. Etwaige Nebeneffekte durch problemirrelevante Eigenschaften der Versuchsumgebung konnten so minimiert bzw. ganz ausgeschlossen werden.

Wie sich in den Messungen am Realsystem [Sch97] zeigte, sind einige Einflüsse, die sich in den Meßergebnissen niederschlagen, begründet in den Protokollmechanismen der Logischen Verbindungsschicht und des verwendeten Übertragungsprotokolls (in diesem Fall NetBIOS), das auf die Dienste dieser Schicht zugreift. Dies wurde in Abschnitt 2 eingehend beschrieben.

Um aussagekräftige, gut zu analysierende Ergebnisse zu erhalten, wurden die zu simulierenden Netzwerksysteme möglichst einfach gehalten. Die Simulationen stellen TokenRing-Netzwerke mit 1 bis 3 Ringen und bis zu 5 miteinander kommunizierenden Rechnerpaaren dar. Die Verbindungsparameter der einzelnen kommunizierenden Prozesse werden innerhalb der Simulationsreihen variiert. So ist es möglich, Einflüsse des verwendeten Übertragungsprotokolls und der verwendeten (simulierten) Hardware sichtbar zu machen.

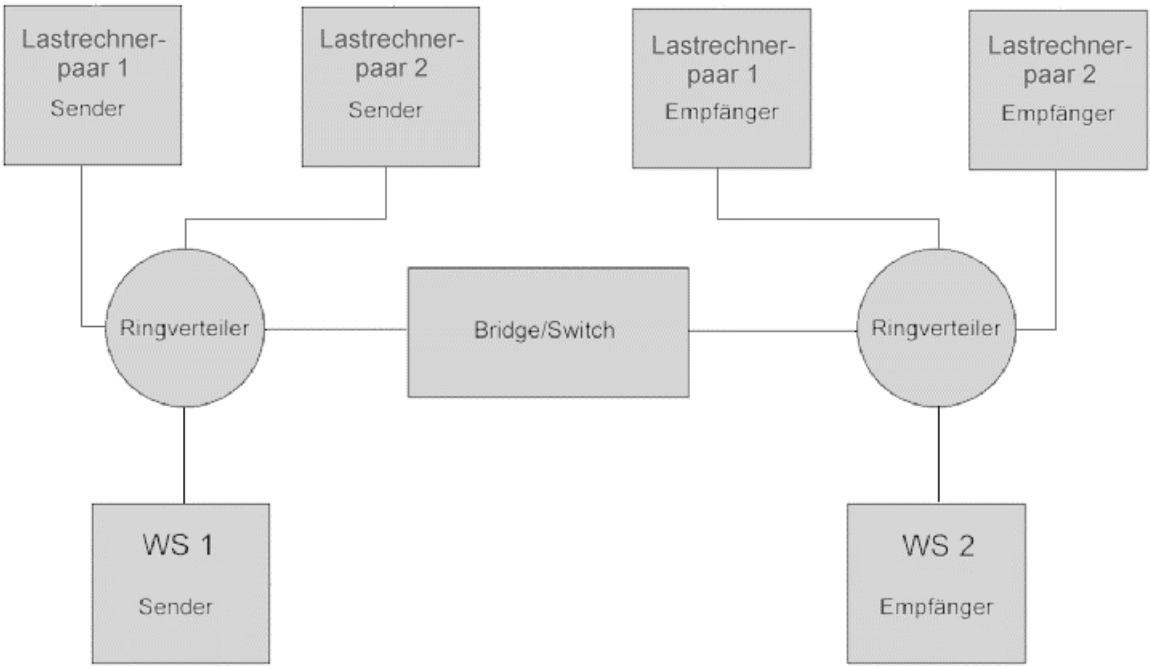


**Abbildung 3.1.1:** Schematische Darstellung eines 2-Ring-Systems mit 2 Lastrechnerpaaren und einem Meßrechnerpaar. Sender und Empfänger der Kommunikationspartner befinden sich stets in verschiedenen Ringen.

## 3.2 Simulationen zum Datendurchsatz

Als Ausgangsbasis und spätere Referenz für die Untersuchungen wurde eine Datenkommunikation zwischen zwei Rechnern, die sich in einem gemeinsamen Ring befinden, simuliert (Abbildung 3.2.1). Diese Variante wird oft auch als Kommunikation über den „Wire“ bezeichnet, da keine zusätzlichen Geräte zwischen die beiden Rechner geschaltet sind und somit eine quasi direkte Verbindung (über einen „Draht“) besteht.

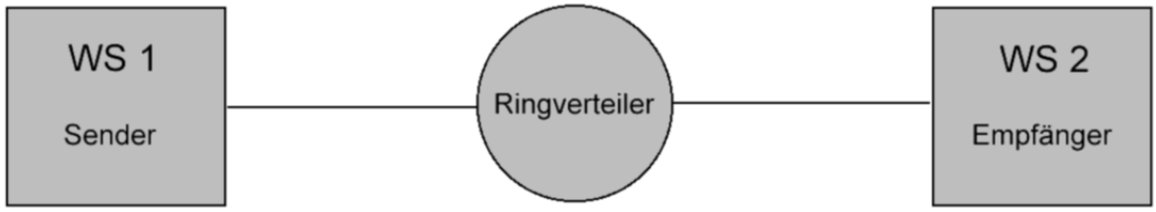


**Abbildung 3.2.1:** Schematische Darstellung eines simulierten 1-Ring-Systems.

Die maximale Datenübertragungsgeschwindigkeit bei einer solchen Konfiguration wird dementsprechend auch „Wire Speed“ genannt. Abbildung 3.2.2 zeigt den Graph (im folgenden auch als Kennlinie bezeichnet) über den Stützstellen gemessenen Simulationswerte für den Datendurchsatz. Als initiale Parameter für die Simulation dienten unter anderem folgenden Werte:

- Maximale Rahmengröße (MTU/MFS): 4200 Byte
- Rahmenfenstergröße (WINDOW): 3
- Paketgröße: 16 - 65535 Byte
- Paket-Generierungswahrscheinlichkeit: 0.42
- Acknowledgement-Generierungswahrscheinlichkeit: 0.6

Die Festlegung der Werte für die Wahrscheinlichkeiten der Generierung von Rahmen/Paketen ist Ergebnis empirischer Untersuchungen. Die Systemparameter des simulierten Netzes hingegen wurden wie beschrieben gewählt, da sie allgemein als

Standardwerte in der Konfiguration von Netzwerken verwendet werden. Ein weiterer Grund liegt in der somit gewährleisteten Vergleichbarkeit der Simulationsergebnisse mit den Ergebnissen aus den Messungen an realen Systemen aus [Sch97].

Das Maximum der gemessenen Werte liegt bei 1.899.966 Byte/s (Paketgröße: 65535 Byte; s. Tabelle 3.2.2) und der Durchschnitt für eine Datenübertragung mit genannter Paketgröße liegt bei 1.899.754 Byte/s, was in etwa den am realen System gemessenen Werten für die Übertragung über den „Wire“ aus [Sch97] entspricht (s. Tabelle 3.2.1). In [Sch97] wurde für diese Konfiguration eine Übertragungsrate von 1.885.840 Byte/s ermittelt. Die Abweichung in den Meßwerten des Datendurchsatzes von simulierter zu realer Datenübertragung beträgt somit in dieser Kategorie nur 13914 Byte/s, was in etwa 0,74 % des real gemessenen Wertes entspricht.

| Paketgröße (Byte) | Durchschnitt der Datendurchsatzmessung für den Wire (Byte/s) | | | Abweichung (ausgehend v. d. Realwerten) | |
|---|---|---|---|---|---|
| | Simulation | | real [Sch97] | in Byte/s | in % |
| 16 | 14246 | (± 0,42) | 14660 | -414 | 2,82 |
| 32 | 28300 | (± 1,16) | 28990 | -690 | 2,38 |
| 64 | 55784 | (± 3,95) | 56523 | -739 | 1,31 |
| 128 | 108571 | (± 8,51) | 110633 | -2062 | 1,86 |
| 256 | 205822 | (± 26,51) | 209836 | -4014 | 1,91 |
| 512 | 373171 | (± 60,48) | 374879 | -1708 | 0,46 |
| 1024 | 628204 | (± 79,98) | 629024 | -820 | 0,13 |
| 2048 | 966544 | (± 128,17) | 936447 | 30097 | 3,21 |
| 4096 | 1290601 | (± 221,28) | 1255379 | 35222 | 2,81 |
| 8192 | 1557573 | (± 181,32) | 1532249 | 25324 | 1,65 |
| 16384 | 1737230 | (± 144,30) | 1702615 | 34615 | 2,03 |
| 32768 | 1839973 | (± 226,59) | 1819362 | 20611 | 1,13 |
| 65535 | 1899754 | (± 115,08) | 1885840 | 13914 | 0,74 |

**Tabelle 3.2.1:** Gegenüberstellung der Meßwerte aus Realmessung und Simulation für den Datendurchsatz in einem 1-Ring-Netzwerksystem (nach Abb. 3.2.1). In Klammern ist der mittlere Fehler für die einzelnen Simulationsreihen aufgeführt.

Die Genauigkeit der Messungen nimmt proportional mit der Menge der übertragenen Daten (Bytes) zu, deshalb sollte eine hinreichend große Datenmenge (Bytes) generiert werden, um aussagekräftige Ergebnisse zu erhalten. Gewählt wurde eine Menge von 500000 Byte zu

sendender Daten bei jeder Messung, da in diesem Fall die Abweichungen in den Meßwerten hinreichend klein und die Durchlaufzeit einer Simulationsreihe akzeptabel sind.

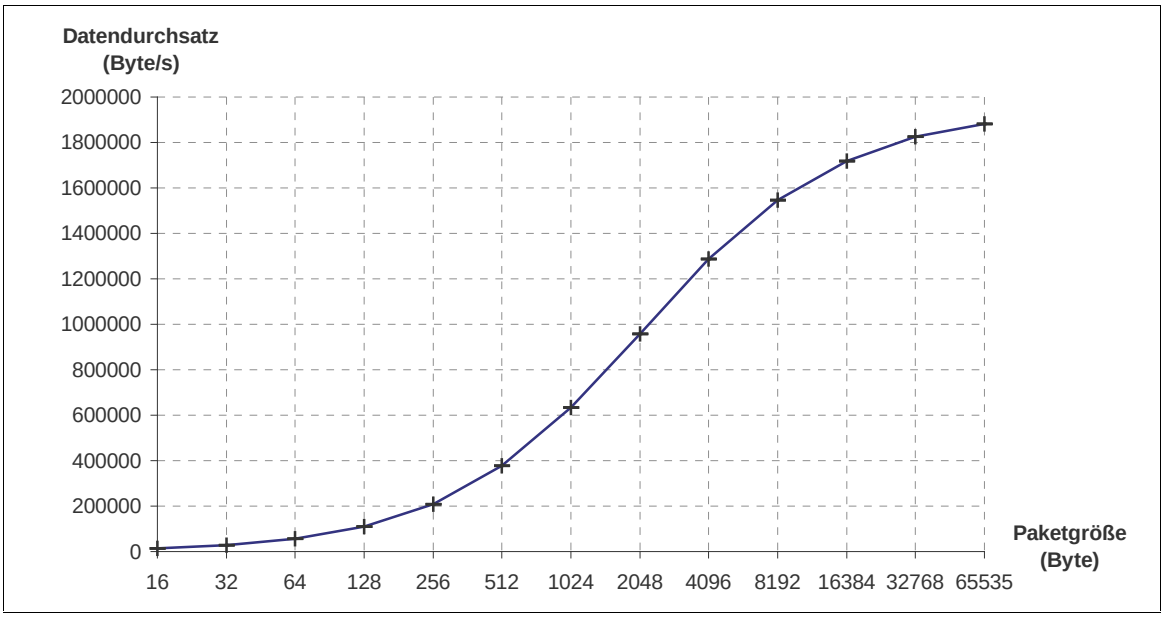


**Abbildung 3.2.2:** Durchschnittswerte für Datendurchsatz aus 10 Simulationsreihen zur Datenübertragung zwischen 2 Rechnern in einem 1-Ring-TR-Netzwerksystem ohne zusätzliche Netzlast (nach Abb. 3.2.1) für den Datendurchsatz.

Es wurden Serien von Simulationsreihen durchgeführt, um anhand der erhaltenen Ergebnisse statistische Betrachtungen durchführen zu können und somit die Möglichkeit zu haben, diese in Beziehung zu den Ergebnissen aus den Messungen am realen System [Sch97] zu setzen.

Die Tabelle 3.2.2 zeigt die gemessenen Werte für ein Rechnerpaar ohne zusätzliche Netzlast. Wie aus den Meßreihen zu erkennen, liegt die mittlere Abweichung der Meßwerte unter 1 %. Der größte Abstand zwischen dem Minimum und dem Maximum der gemessenen Werte zwischen den einzelnen Meßreihen (bei jeweils gleichen Ausgangsparametern für die Simulation) wurde mit 778 Byte/s, bei einer eingestellten Paketgröße von 4200 Byte, festgestellt. Dies entspricht ≈ 0,06 % des mittleren gemessenen Datendurchsatzes. Auf Grund dieser geringen Abweichungen kann davon ausgegangen werden, daß im Regelfall schon eine Meßreihe mit dem Simulator aussagekräftige Ergebnisse liefert.

| Meßreihe | #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 | #9 | #10 |
|---|---|---|---|---|---|---|---|---|---|---|
| **16** | 14246 | 14246 | 14246 | 14246 | 14245 | 14246 | 14246 | 14246 | 14245 | 14246 |
| **32** | 28301 | 28301 | 28300 | 28301 | 28300 | 28301 | 28299 | 28302 | 28300 | 28298 |
| **64** | 55781 | 55788 | 55784 | 55790 | 55785 | 55779 | 55784 | 55781 | 55781 | 55790 |
| **128** | 108566 | 108555 | 108569 | 108567 | 108569 | 108583 | 108573 | 108567 | 108581 | 108585 |
| **256** | 205809 | 205822 | 205777 | 205804 | 205836 | 205845 | 205798 | 205851 | 205854 | 205828 |
| **512** | 373203 | 373116 | 373243 | 373229 | 373116 | 373054 | 373168 | 373161 | 373227 | 373190 |
| **1024** | 628101 | 628237 | 628114 | 628320 | 628216 | 628186 | 628272 | 628310 | 628151 | 628132 |
| **2048** | 966504 | 966538 | 966510 | 966676 | 966735 | 966364 | 966487 | 966673 | 966355 | 966600 |
| **4096** | 1290431 | 1290745 | 1291005 | 1290591 | 1290700 | 1290720 | 1290640 | 1290596 | 1290351 | 1290227 |
| **8192** | 1557553 | 1557350 | 1557488 | 1557619 | 1557336 | 1557858 | 1557568 | 1557880 | 1557575 | 1557503 |
| **16384** | 1737329 | 1737239 | 1736950 | 1737347 | 1737347 | 1737175 | 1737230 | 1737211 | 1737049 | 1737419 |
| **32768** | 1839826 | 1840090 | 1840282 | 1840090 | 1840110 | 1839715 | 1840302 | 1839796 | 1839796 | 1839725 |
| **65535** | 1899705 | 1899618 | 1899727 | 1899694 | 1899629 | 1899716 | 1899966 | 1899868 | 1899727 | 1899890 |

**Tabelle 3.2.2 :** Datendurchsatz-Meßreihen #1 - #10 der Simulation einer Kommunikation zweier Rechner innerhalb eines Ringes bei Paketgrößen von 16 – 65535 Byte und einer maximalen Rahmengröße von 4200 Byte.

Wird der Ring zusätzlich durch andere laufende Kommunikationen belastet, so sinkt der Datendurchsatz erwartungsgemäß (Abbildung 3.2.3). Bei Verwendung von kleineren Paketen ist der zusätzliche Datenverkehr auf dem Ring von wesentlich geringerer Bedeutung als bei Verwendung von mittleren bzw. großen Paketen. Dies ist leicht nachvollziehbar, da sich die Wahrscheinlichkeit von Konkurrenzen um ein freies Token mit der Größe der gesendeten Rahmen und somit der Ringauslastung verändert.

| Paketgröße | kein Lastrechner | 1 Lastrechnerpaar | 2 Lastrechnerpaare |
|---|---|---|---|
| **16** | 14246 | 13781 | 13321 |
| **32** | 28300 | 27206 | 26145 |
| **64** | 55784 | 52918 | 50249 |
| **128** | 108571 | 100572 | 93540 |
| **256** | 205822 | 182556 | 164048 |
| **512** | 373171 | 308624 | 262840 |
| **1024** | 628204 | 474095 | 378298 |
| **2048** | 966544 | 649046 | 485228 |
| **4096** | 1290601 | 778224 | 558246 |
| **8192** | 1557573 | 872579 | 613582 |
| **16384** | 1737230 | 937407 | 641920 |
| **32768** | 1839973 | 951875 | 649243 |
| **65535** | 1899754 | 967664 | 650653 |

**Tabelle 3.2.3:** Datendurchsatz (Simulation eines 1-Ring-Netzwerksystems ); Angaben in Byte/s; bei Paketgrößen von 16 – 65535 Byte und bei veränderter Netzlast (0 – 2 Lastrechnerpaare).

Die Tabelle 3.2.3 in Verbindung mit Abbildung 3.2.3 zeigt die Werte aus Simulationsreihen mit und ohne zusätzliche Netzlast. Bei einer Paketgröße von 65535 Byte ist besonders deutlich zu erkennen, daß die Übertragungskapazität des Ringes schon ohne Netzlast

nahezu ausgeschöpft wird. Werden Netzlasten gleicher Art und Größe wie die Primärübertragung eingeführt, so teilen sich die konkurrierenden Kommunikationen die Ringkapazität gleichmäßig auf. Das setzt aber voraus, daß alle Rechnerpaare die gleichen Leistungsparameter besitzen und kein Rechner im Ring das Token bevorzugt erhält (Prioritätssteuerung).

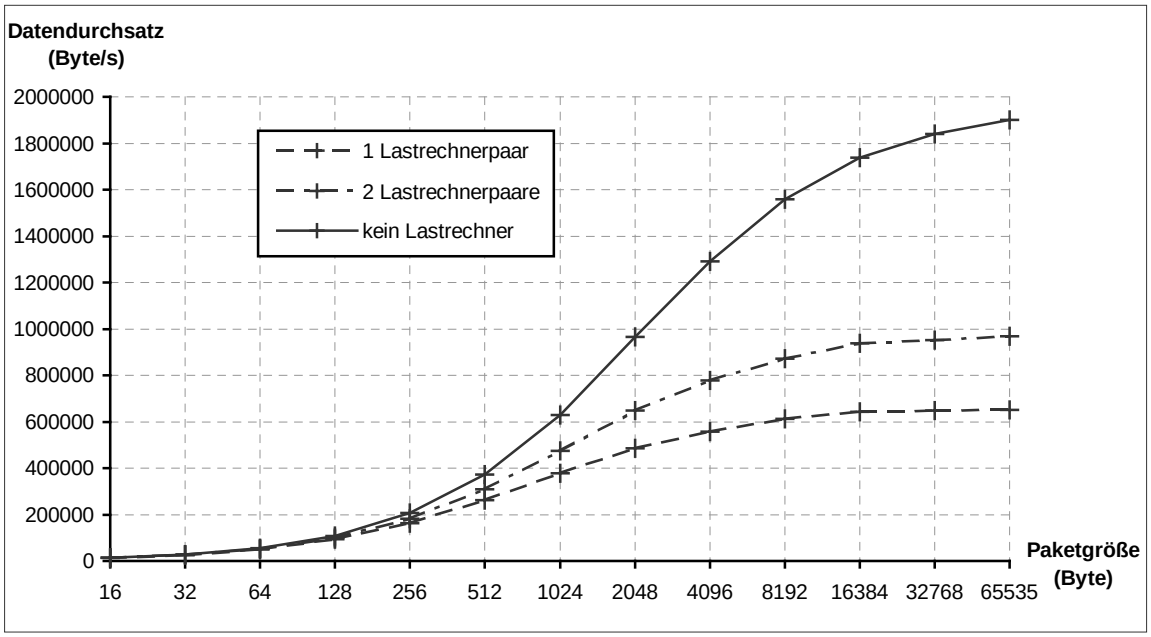


**Abbildung 3.2.3:** Datendurchsatz (Simulation eines 1-Ring-Systems mit 0 – 2 Lastrechnerpaaren) bei einer maximalen Rahmengröße von 4200 Byte und Paketgrößen von 16 – 65535 Byte.

Aus den Werten in Tab. 3.2.3 geht ebenfalls hervor, daß mit der steigenden Anzahl der aktiven Verbindungen auf dem Ring das Problem des Overhead mehr und mehr an Bedeutung gewinnt. Es wird aber auch deutlich, daß die Ringkapazität bei mehreren parallelen Verbindungen besser ausgenutzt wird. Während eine einzelne Datenverbindung den Ring nur zu 90,6 % auslastet, erreichen zwei gleichzeitig ablaufende Übertragungen eine Auslastung von 92,3 % und drei Verbindungen sogar 93,1 % der Ringkapazität.

### 3.2.1 Store and Forward

Arbeitet ein Ringkopplungsgerät (Bridge/Switch) im „Store and Forward“ („Speichern und Weiterreichen“) –Modus, so werden die zu übertragenden Rahmen erst komplett von dem

Gerät aus dem Quellring heraus empfangen, bevor sie in den Zielring weitergeleitet werden (siehe dazu Abb. 3.2.6).

Bei Übertragungen im Store & Forward - Modus wird die Abhängigkeit zwischen Protokollmechanismen und Datendurchsatz besonders deutlich. Ein Indiz hierfür ist der unterschiedliche Kurvenverlauf bei veränderten protokollabhängigen Kommunikationsabläufen.

Die Abbildung 3.2.3 zeigt den am Simulator gemessenen Datendurchsatz für ein TokenRing-Netzwerk bestehend aus zwei Ringen, die durch ein Ringkopplungsgerät (Bridge/Switch) miteinander verbunden sind (nach Abb. 3.2.4). Das Ringkopplungsgerät arbeitet im Store & Forward-Modus. Die maximale Rahmengröße auf dem Ring (*maximum transferable unit* – MTU; *maximum frame size* - MFS) wurde auf 2200 Byte (Strich-Punkt-Linie), 4200 Byte (durchgehende Linie) bzw. 17000 Byte (gepunktete Linie) festgelegt. Die Übertragung wurde für Paketgrößen von 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65535 Byte simuliert. Für die Meßreihe mit der MFS von 4200 Byte (durchgehende Linie) wurden zusätzlich Simulationen in den Intervallen von (a) 2048 – 8192 Byte Paketgröße in 64 Byte-Schritten und (b) 4195 – 4225 Byte in 1 Byte-Schritten durchgeführt.

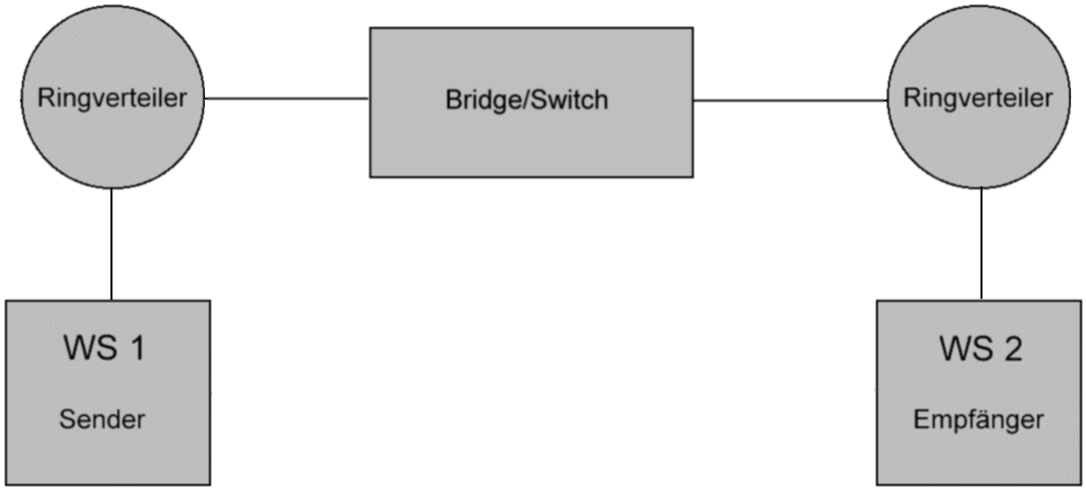


**Abbildung 3.2.4:** Schematische Darstellung eines 2-Ring-Netzwerksystems, verbunden mittels Bridge/Switch. Sender und Empfänger befinden sich in verschiedenen Ringen. WS 1 befindet sich also im Quellring und WS 2 im Zielring der Datenübertragung.

Alle Graphen in Abb. 3.2.5 zeigen das gleiche Verhalten bei Erreichen der maximal übertragbaren Rahmengröße (MFS/MTU) durch die zur Übertragung benutzten Paketgröße. Durch die unter (a) und (b) gemachten Messungen wurden keine weiteren

Unregelmäßigkeiten sichtbar. Die aufgetretenen Abweichungen blieben innerhalb der Toleranzwerte für unterschiedliche Simulationsdurchläufe. Der Anstieg des Graphen für den Datendurchsatz wird unterhalb der MFS, mit Näherung an selbige, stetig flacher. Überschreitet die Paketgröße die maximale Rahmengröße auf dem Ring, so nehmen die Werte für den Datendurchsatz wieder stärker zu. Dieses Verhalten ist erklärbar durch die sich in den beiden Bereichen oberhalb und unterhalb des Wertes für die MTU-Größe unterschiedlich auswirkenden Protokollmechanismen.

Das Protokoll der Verbindungsschicht wirkt sich in zweierlei Hinsicht unterschiedlich auf die Übertragung aus. Zum einen ändert sich beim Überschreiten der

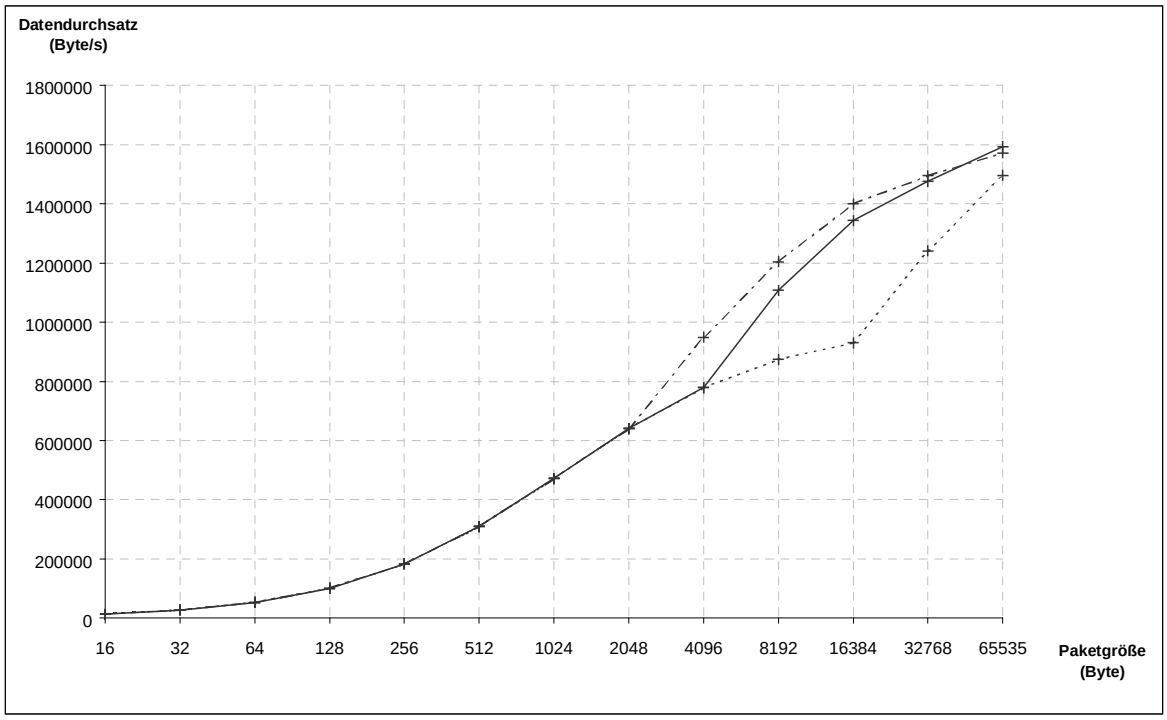


**Abbildung 3.2.5:** Datendurchsatz (Simulation) für Übertragung über Bridge/Switch im Store & Forward-Modus. Die jeweiligen Werte für die maximale Rahmengröße (MFS/MTU) betragen 2200 Byte (Strich-Punkt-Linie) 4200 Byte (durchgehende Linie) und 17000 Byte (gepunktete Linie)

maximalen Rahmengröße das Verhältnis zwischen Daten- und Kontrollrahmen.

Ist *n* die Anzahl der Datenrahmen pro Datenpaket, so ist das Verhältnis von Daten- zu Kontrollrahmen pro Paket $n : (n + 1)$. Auf einen gesendeten Daterahmen kommen jeweils ein Rahmen mit der Sequenznummer des übermittelten Datenrahmens und ein Antwortrahmen (engl.: *acknowledgement frame*; *ACK-Frame*) als Quittung für ein komplett empfangenes Datenpaket.

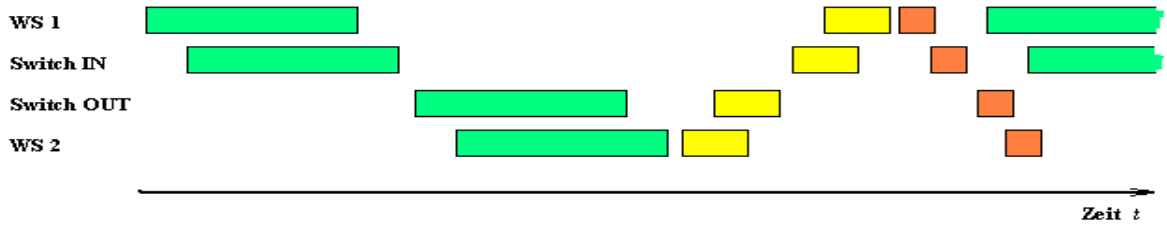


**Abbildung 3.2.6:** Zeitlicher Ablauf der Kommunikation zwischen Client- und Serverrechner in einem 2-Ring-System über eine Bridge im Store & Forward - Modus bei Paketgrößen im Bereich unterhalb der benutzten MTU-Größe. Die Farben der Balken sind folgendermaßen zugeordnet: grün – Datenrahmen, gelb – LLC-Sequenzrahmen, rot – NetBIOS-Acknowledgement.

Ist das Paket kleiner oder gleich der maximal übertragbaren Rahmengröße (abzüglich Header und Trailer des Rahmens), so ist das Verhältnis stets 1:2. Überschreitet die Paketgröße die MTU-Größe, so ändert sich dieses Verhältnis proportional zur Anzahl der zu sendenden Rahmen pro Paket (2:3, 3:4, 4:5, …, $n$:($n$ + 1)). Da sich das Verhältnis kontinuierlich zugunsten der Datenrahmen verschiebt, steigt das Übertragungspotential der Verbindung und somit auch der Datendurchsatz wieder stärker an. Im weiteren Verlauf nimmt der Einfluß der Veränderung dieses Verhältnisses immer mehr ab und das Übertragungspotential der Verbindung nähert sich, bei Verwendung der vorgegebenen Protokolle dem Sättigungswert, der maximalen Übertragungsrate.

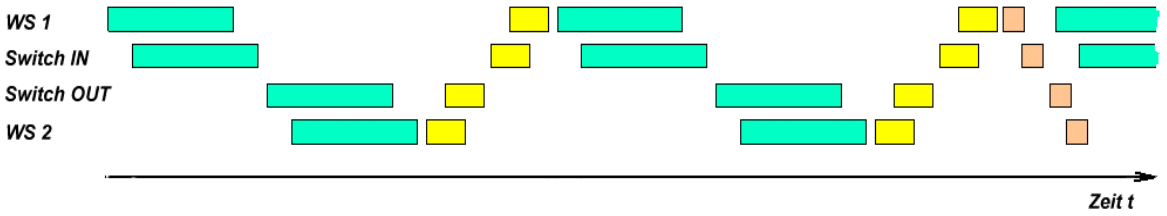


**Abbildung 3.2.7:** Zeitlicher Verlauf der Kommunikation zwischen Client- und Serverrechner bei Paketgrößen im Bereich oberhalb der festgelegten MTU-Größe (hier bei Verwendung von Store & Forward). Die Farben der Balken sind folgendermaßen zugeordnet: grün – Datenrahmen, gelb – LLC-Sequenzrahmen, rot – NetBIOS-Acknowledgement.

Auf der anderen Seite wird durch das Aufteilen der Pakete in mehrere Rahmen die Ausnutzung des Fenstermechanismus möglich, da dieser nur über den zu jeweils ein und demselben Paket gehörenden Datenrahmen wirksam ist [RFC02]. Die Fenstergröße richtet sich nach den Systemeinstellungen beider Kommunikationspartner (maximale Fenstergröße) und der Qualität der Verbindung (Signalverzögerung, Fehlerrate). Sie wird während des Datenaustausches zum Aufbau der Verbindung festgelegt (siehe Abschnitt 2).

| Meßreihe | #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 | #9 | #10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 16 | 13784 | 13784 | 13784 | 13784 | 13783 | 13783 | 13784 | 13783 | 13783 | 13783 |
| 32 | 27213 | 27211 | 27211 | 27212 | 27212 | 27212 | 27212 | 27213 | 27211 | 27212 |
| 64 | 52920 | 52917 | 52927 | 52923 | 52923 | 52924 | 52920 | 52927 | 52924 | 52925 |
| 128 | 100560 | 100562 | 100568 | 100560 | 100572 | 100569 | 100563 | 100576 | 100553 | 100566 |
| 256 | 182472 | 182428 | 182442 | 182433 | 182450 | 182423 | 182425 | 182436 | 182454 | 182432 |
| 512 | 308378 | 308353 | 308328 | 308343 | 308353 | 308346 | 308373 | 308375 | 308358 | 308363 |
| 1024 | 470135 | 470074 | 470064 | 469975 | 470085 | 470163 | 470043 | 470183 | 470113 | 470125 |
| 2048 | 637841 | 637904 | 637936 | 637771 | 637793 | 637751 | 637895 | 637823 | 637929 | 637850 |
| 4096 | 775941 | 776074 | 776036 | 776098 | 776026 | 776089 | 776065 | 776134 | 775922 | 776056 |
| 8192 | 1104045 | 1104239 | 1103957 | 1104089 | 1103983 | 1104005 | 1104067 | 1104089 | 1104151 | 1104079 |
| 16384 | 1192709 | 1192687 | 1192598 | 1192807 | 1187277 | 1192824 | 1187230 | 1192696 | 1192743 | 1190628 |
| 32768 | 1235584 | 1235419 | 1240927 | 1235594 | 1241056 | 1235571 | 1240862 | 1235419 | 1240913 | 1239691 |
| 65535 | 1267379 | 1267404 | 1267083 | 1272071 | 1267408 | 1267170 | 1272071 | 1271909 | 1267399 | 1270433 |

**Tabelle 3.2.4 :** Datendurchsatz-Meßreihen #1 - #10 der Simulation einer Kommunikation zweier Rechner über eine Bridge bzw. einen Switch (Store & Forward) bei Paketgrößen von 16 – 65535 Byte

Aus den Schemata für den zeitlichen Ablauf einer Übertragung mit Paketgrößen unterhalb des Wertes für die MTU (Abb. 3.2.6) und Paketgrößen oberhalb der MTU (Abb. 3.2.7) wird deutlich, woraus die Sättigung des Übertragungspotentials in den verschiedenen Abschnitten der Graphen resultiert. Es ist erkennbar, daß es einer einzelnen Verbindung,

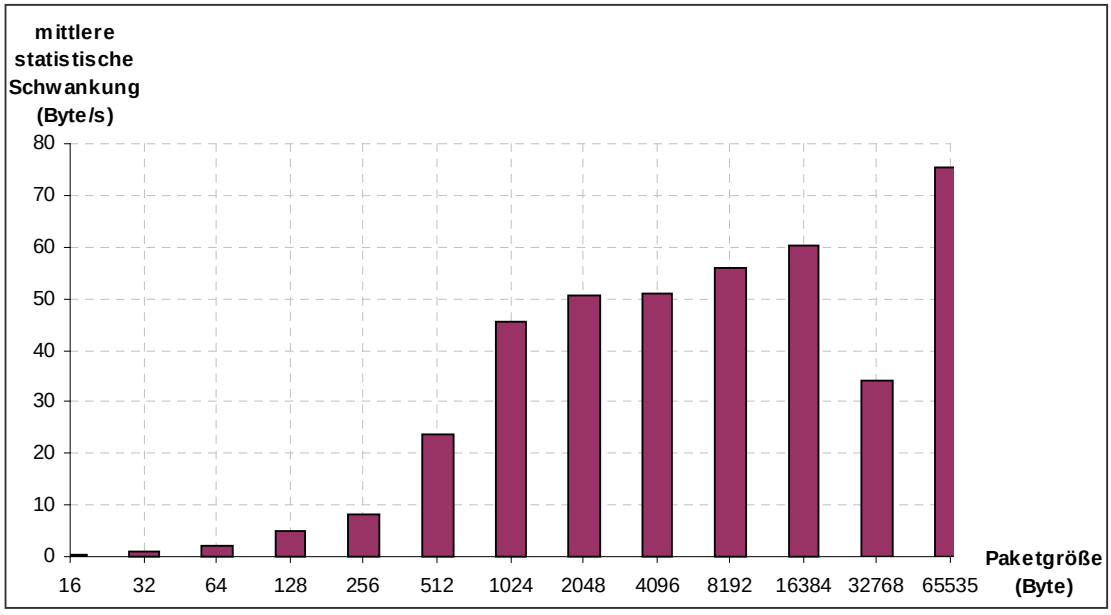


**Abbildung 3.2.8:** Mittlere statistische Schwankung für Datendurchsatzmeßreihen; Simulation eines 2-Ring-Systems im Store & Forward - Modus, bei Verwendung einer maximalen Rahmengröße von 4200 Byte und Paketgrößen laut Beschriftung.

ohne Nutzung des Fenstermechanismus, nicht möglich ist, die volle zur Verfügung stehende Bandbreite auszunutzen (je nach verwendeter Paket- und Rahmengröße bis maximal ca

50 %). Wird der durch das Protokoll der Verbindungsschicht bereitgestellte Fenstermechanismus genutzt, d.h. auf Sender- wie auch auf Empfängerseite ein Wert >1 für die maximale Fenstergröße eingestellt, so kann eine Ausnutzung der Bandbreite von bis zu ca. 80 % erreicht werden (vergl. Abb. 3.2.5).
Der Overhead bei Verwendung von Datenpaketen mit einer Größe unterhalb von 1KByte ist relativ hoch (32 Byte Kopfdaten + 32 Antwortrahmen + 18 LLC-Sequenzrahmen = 82 Byte, das entspricht ≈ 7,41 % der Gesamtdatenmenge von 1106 Byte pro Paket). Hinzu kommt die zeitliche Verzögerung durch Rahmengenerierung und Übertragung. Somit sind kleine Paketgrößen für Übertragungen von großen Datenmengen im allgemeinen relativ ungeeignet.

Befinden sich mehrere (in diesem Fall 1 – 4) gleichzeitig miteinander kommunizierende Rechnerpaare im System (2-Ring-System), so teilen sich diese wie in der Simulation eines 1-Ring-Systems, die zur Verfügung stehende Bandbreite zu annähernd gleichen Teilen, vorausgesetzt alle Rechner arbeiten mit den gleichen System-, Leistungs- und Verbindungsparametern.

| ***Paketgröße*** | ***Kein Lastrechnerpaar*** | ***1 Lastrechnerpaar*** | ***2 Lastrechnerpaare*** | ***3 Lastrechnerpaare*** |
|---|---|---|---|---|
| **16** | 13837 | 13661 | 13268 | 12852 |
| **32** | 27316 | 26812 | 26000 | 25076 |
| **64** | 53129 | 51482 | 49725 | 47615 |
| **128** | 100958 | 95651 | 91816 | 86823 |
| **256** | 183157 | 167016 | 158073 | 146801 |
| **512** | 309588 | 267122 | 249978 | 224218 |
| **1024** | 471990 | 380696 | 349658 | 308384 |
| **2048** | 640372 | 484189 | 440963 | 378191 |
| **4096** | 778935 | 560557 | 498115 | 429923 |
| **8192** | 1108140 | 712119 | 593495 | 473602 |
| **16384** | 1350916 | 867542 | 632804 | 483890 |
| **32768** | 1446790 | 900530 | 647730 | 482000 |
| **65535** | 1576355 | 958026 | 657340 | 494026 |

**Tabelle 3.2.5:** Datendurchsatz für Übertragung von WS1 zu WS2 (Simulation) in Byte/s; 2-Ring-System (nach Abb. 3.2.4) bei unterschiedlichen Netzlasten und Paketgrößen (angegeben in Byte) im Store & Forward-Modus;

Bei Verwendung von Datenpaketen mit geringer Größe (16 – 256 Byte) zur Übertragung ist der Performance-Verlust, hervorgerufen durch die geringere Bandbreite pro Kommunikationsweg, weitaus weniger spürbar als bei Verwendung von großen

Datenpaketen. Bei Paketgrößen bis 256 Byte ist der Performance-Verlust geringer als 10 % gegenüber dem nächst niederen Lastaufkommen. Demgegenüber liegt der Verlust bei Verwendung von Paketgrößen von 64 KByte zur Übertragung bei nahezu 40 %. Die Tabelle 3.2.5 enthält die gemessenen Werte der Simulation, unterteilt nach Anzahl der Lastrechnerpaare im System und der jeweils verwendeten Paketgröße. Die maximale Rahmengröße ist für die durchgeführten Simulationen auf 4200 Byte festgelegt worden, was dem Standardwert in realen TokenRing-Systemen entspricht. Die Rahmenfenstergröße ist 3, d.h. drei Datenrahmen dürfen nacheinander gesendet werden, ohne die Bestätigung des Empfängers abwarten zu müssen (siehe Abschnitt 2).

### 3.2.2 Cut-Through-Modus

Der „Cut-Through“ (dtsch. etwa: „Durchschieben“) –Modus ist charakterisiert durch das sofortige Weiterleiten der Rahmen, sobald durch Adressauflösung der Zielring festgestellt wurde. Der zu übertragende Rahmen muß also nicht erst komplett empfangen werden, bevor er zum Zielring weitergeleitet wird (vergl. Abbildungen 3.2.9 vs. Abbildung 3.2.6). Da die einzelnen Rahmen nicht mehr zwischengespeichert werden müssen, sinken die Latency und die Speicherbelastung der Netzkopplungsgeräte.
Unter Latency (dtsch.: Latenzzeit) versteht man die Zeitverzögerung, die ein Rahmen beim Durchlaufen eines Gerätes (Bridge/Switch) erfährt.

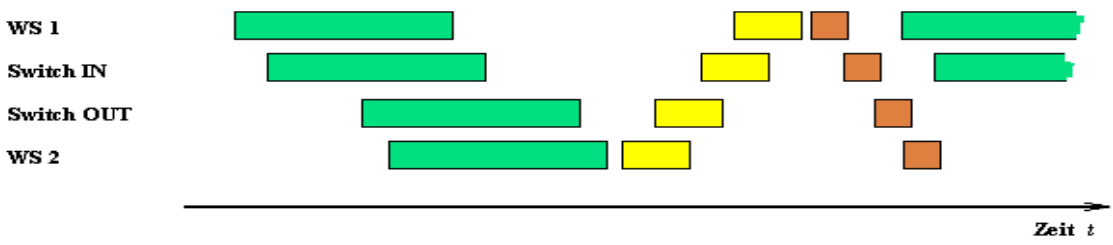


**Abbildung 3.2.9:** Zeitlicher Verlauf einer Datenübertragung im Cut-Through-Modus über eine Typ 2-Verbindung. Die Fenstergröße beträgt in diesem Fall 1. Die Paketgröße liegt unterhalb der maximalen Rahmengröße. Erkennbar ist die bessere Ausnutzung der Bandbreite gegenüber einer Übertragung im Store&Forward-Modus. Die Farben der Balken sind folgendermaßen zugeordnet: grün – Datenrahmen, gelb – LLC-Sequenzrahmen, rot – NetBIOS-Acknowledgement.

Je größer die zur Übertragung verwendeten Datenrahmen sind, desto größer fällt auch der Unterschied in den Latenzzeiten zwischen Netzkopplungsgeräten aus, die in unterschiedlichen Übertragungsmodi (Store & Forward vs. CutThrough) arbeiten. Gleiches gilt hinsichtlich der Speicherbelastung dieser Geräte.

| Sender / Empfänger | Rahmentyp | Rahmengröße (in Byte) | Bemerkung |
|---|---|---|---|
| WS1-> WS2 | Data First Mid | 4200 | 1. Datenrahmen |
| WS1-> WS2 | Data First Mid | 4200 | 2. Datenrahmen |
| WS2-> WS1 | LLC | 18 | Bestätigung des 1. Datenrahmens |
| WS1-> WS2 | Data First Mid | 4200 | 3. Datenrahmen |
| WS2-> WS1 | LLC | 18 | Bestätigung des 2. Datenrahmens |
| WS1-> WS2 | Data Only Last | 3868 | letzter Datenrahmen des Paketes |
| WS2-> WS1 | LLC | 18 | Bestätigung des 3. Datenrahmens |
| WS2-> WS1 | ACK | 32 | Bestätigung des Paketes |
| WS1-> WS2 | LLC | 18 | Bestätigung des ACK-Rahmens und Nummerierung des Paketes |
| WS1-> WS2 | Data First Mid | 4200 | Übertragung des nächsten Paketes |
| WS1-> WS2 | Data First Mid | 4200 | |
| WS2-> WS1 | LLC | 18 | |

**Tabelle 3.2.6:** Mitschnitt einer simulierten Datenübertragung von WS1 nach WS2 über eine Brücke bei einer maximalen Rahmengröße von 4200 Byte und einer Paketgröße von 16384 Byte.

Wird zusätzlich der Fenstermechanismus (*Windowing*), den die Verbindungsschicht bereitstellt, genutzt, so verstärkt sich dieser Effekt entsprechend. Wie schon erwähnt ist hierfür Voraussetzung, daß die verwendete Paketgröße die aktuelle MFS/MTU überschreitet (siehe Abbildung 3.2.10). Im Idealfalle ist das Paket um ein Vielfaches größer als die Rahmengröße. Auf diese Weise wird der Overhead pro gesendetes Datenpaket minimiert. Das Windowing führt durch die Reduzierung von Wartezeiten (Warten auf Bestätigungsrahmen) dazu, daß die Kapazität der Ringe besser ausgenutzt wird, was wiederum zu einer höheren Übertragungsrate für die entsprechende Verbindung führt.

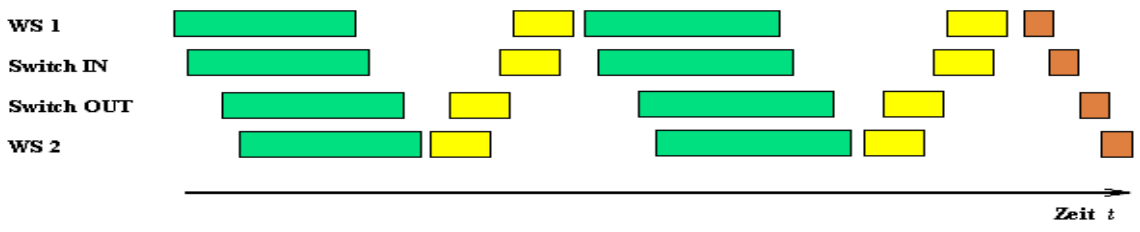


**Abbildung 3.2.10:** Schema einer Übertragung eines Datenpaketes über einen Switch bzw. eine Bridge im CutThrough-Modus bei Paketgrößen oberhalb der MFS/MTU-Größe. Die Farben der Balken sind folgendermaßen zugeordnet: grün – Datenrahmen, gelb – LLC-Sequenzrahmen, rot – NetBIOS-Acknowledgement.

Die Abbildung 3.2.11 macht deutlich, wie sich dieser Mechanismus (Fenstergröße: 2 Rahmen) auf die Bandbreitenausnutzung auswirkt. *WS 1* sendet nacheinander zwei zu einem Datenpaket gehörende Rahmen an *WS 2*, ohne auf die Empfangsbestätigung des ersten Rahmens durch *WS 2* zu warten. Der Leerlauf der Rechnersysteme und des Ringes werden verringert und so eine höhere Datendurchsatzrate für diese Verbindung erzielt.

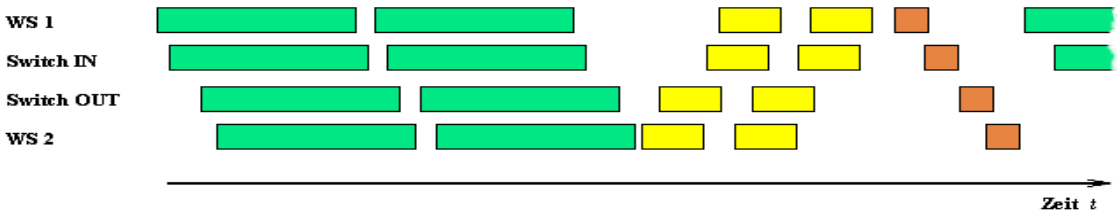


**Abbildung 3.2.11:** Schema einer Übertragung eines Datenpaketes über einen Switch bzw. eine Bridge im CutThrough-Modus bei Paketgrößen oberhalb der MFS/MTU-Größe und einer Rahmenfenstergröße von 2 Rahmen. Die Farben der Balken sind folgendermaßen zugeordnet: grün – Datenrahmen, gelb – LLC-Sequenzrahmen, rot – NetBIOS-Acknowledgement.

Besonders bei Übertragungen über mehrere Ringe hinweg macht sich diese Eigenschaft positiv bemerkbar, da auf diese Weise die durch die längeren Übertragungswege und durch die verschiedenen zu durchlaufenden Netzkopplungsgeräte hervorgerufenen Verzögerungen relativiert werden können.

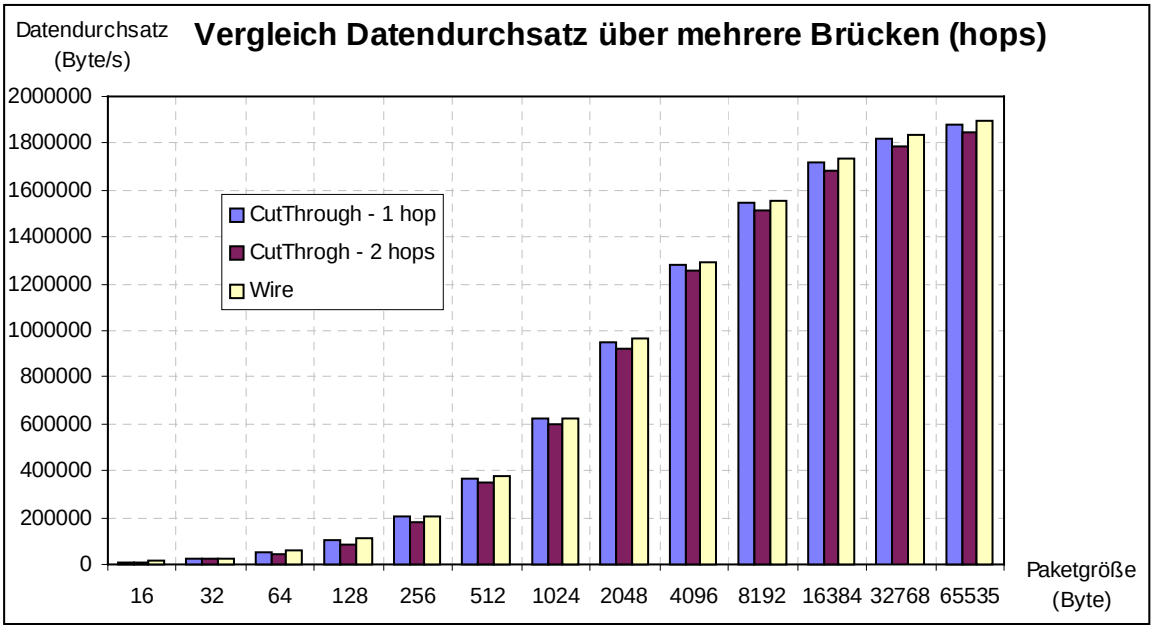


**Abbildung 3.2.12 :** Datendurchsatzmessungen (Simulation) , Cut-Through-Modus, 0 - 2 Brücken (*hops*), MTU = 4200 Byte, Fenstergröße = 3 Rahmen

Die Abbildung 3.2.12 zeigt einen Vergleich der Werte für den Datendurchsatz aus verschiedenen Simulationen zur Datenübertragung im CutThrough-Modus bei verschiedenen Paketgrößen über eine Brücke (blauer Balken), über 2 Brücken (brauner Balken) und, als Referenz, die Werte einer Übertragung (ebenfalls Simulation) innerhalb eines Ringes („*Wire*“; gelber Balken). Es wurden keine Lastrechnerpaare in die Simulation einbezogen. Auf diese Weise werden die Werte für den Performance-Verlust aufgrund des Bridging („Daten über eine Bridge leiten“) nicht durch etwaige Wartezeiten auf ein freies Token in den entsprechenden Ringen verfälscht.

| *Meßreihe* | *#1* | *#2* | *#3* | *#4* | *#5* | *#6* | *#7* | *#8* | *#9* | *#10* |
|---|---|---|---|---|---|---|---|---|---|---|
| **16** | 12732 | 12732 | 12732 | 12732 | 12732 | 12732 | 12732 | 12732 | 12732 | 12732 |
| **32** | 25464 | 25464 | 25464 | 25464 | 25464 | 25464 | 25464 | 25464 | 25464 | 25464 |
| **64** | 50929 | 50929 | 50929 | 50929 | 50929 | 50929 | 50929 | 50929 | 50929 | 50929 |
| **128** | 101859 | 101860 | 101859 | 101859 | 101859 | 101859 | 101859 | 101859 | 101859 | 101859 |
| **256** | 202548 | 202575 | 202548 | 202549 | 202549 | 202550 | 202526 | 202537 | 202526 | 202540 |
| **512** | 367580 | 367725 | 367729 | 367719 | 367537 | 367716 | 367666 | 367712 | 367697 | 367781 |
| **1024** | 620160 | 619983 | 620106 | 620186 | 620001 | 620033 | 619965 | 620019 | 620170 | 619913 |
| **2048** | 944869 | 944602 | 944584 | 944546 | 944961 | 944710 | 944861 | 944675 | 944934 | 944681 |
| **4096** | 1279365 | 1279394 | 1279375 | 1279507 | 1279370 | 1279714 | 1279891 | 1279503 | 1279532 | 1279232 |
| **8192** | 1538138 | 1537910 | 1537541 | 1537917 | 1537654 | 1536561 | 1536873 | 1536746 | 1536817 | 1536405 |
| **16384** | 1716275 | 1716178 | 1716735 | 1716098 | 1716558 | 1716098 | 1717603 | 1716399 | 1716540 | 1717098 |
| **32768** | 1819024 | 1817971 | 1818140 | 1818219 | 1816989 | 1818150 | 1818378 | 1817802 | 1817475 | 1818259 |
| **65535** | 1876356 | 1876580 | 1876207 | 1875962 | 1876740 | 1877358 | 1877369 | 1877060 | 1876367 | 1876537 |

**Tabelle 3.2.7 :** Meßreihen für Datendurchsatz (Simulation) eines 2-Ring-System (Cut-Through-Modus) nach Abb. 3.2.2

## 3.3 Simulationen zur Speicherbelastung

Von Interesse für eine optimale konzeptionelle Erarbeitung oder Umstrukturierung eines Netzwerkes ist auch, inwieweit die Speicherkapazität der Netzkopplungsgeräte Einfluß auf ihre Übertragungseigenschaften besitzt.

Aus Realsystemen ist bekannt, daß in Extremsituationen (Auftreten von Verbindungsfehlern) die Geräte überlastet werden können. Als Beispiele wären hier Broadcaststürme (durch Erneutes Hochfahren des Systems nach Stromausfall) oder das Auftreten des sogenannten Beaconing (kann durch Kabelschäden auftreten) zu nennen.

In diesem Abschnitt sollen nun die Auswirkungen der Belastung des Netzwerkes durch verschiedenartige Datenverkehrsaufkommen auf die Anforderungen an die Speicherkapazität der Netzkopplungsgeräte anhand einer Simulation untersucht werden. Um aussagekräftige Ergebnisse bezüglich der Dynamik der Speicherbelastung bei veränderten äußeren Bedingungen zu gewinnen, wurde wiederum darauf geachtet, das via Simulation nachzubildende System, so einfach wie möglich zu halten.

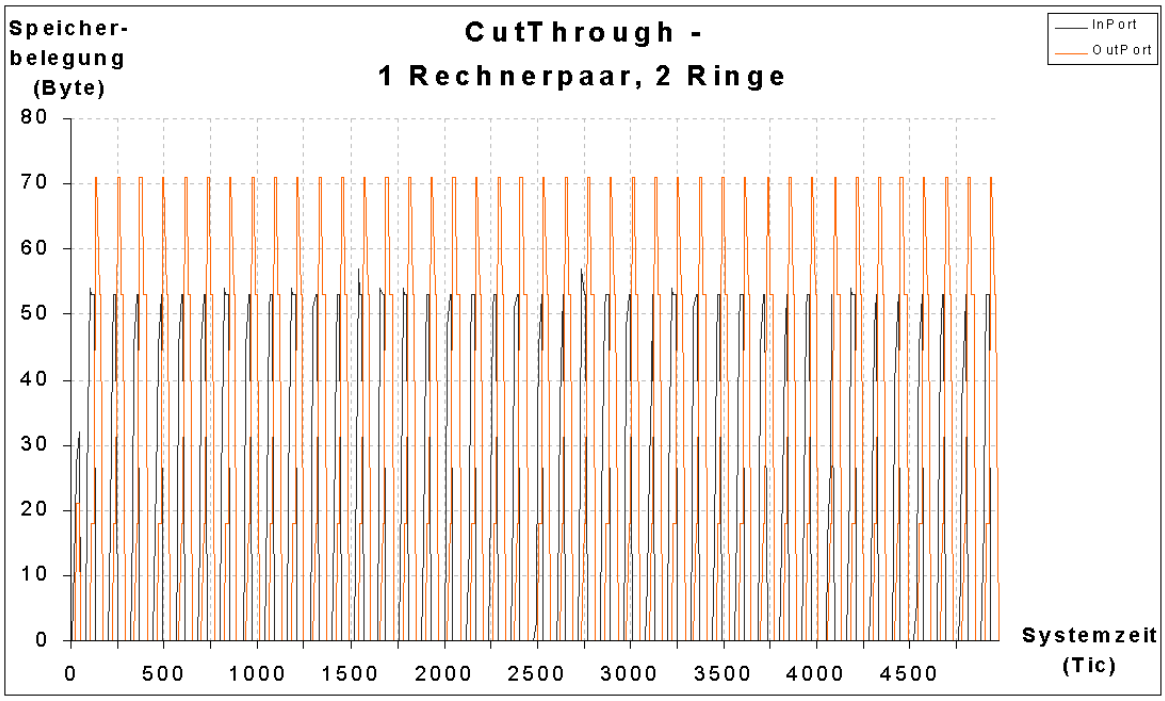


**Abbildung 3.3.1 :** Simulationsmeßwerte der Speicherbelastung der Eingangs- und Ausgangsports einer Bridge die zwei Ringe miteinader verbindet und über die zwei Rechner miteinander kommunizieren. Größe der übertragenen Datenpakete – 32 Byte; MTU – 4200 Byte; Window – 2; Übertragungsmodus – CutThrough.

Das simulierte Netzwerk besteht aus zwei Ringen die mittels einer Bridge miteinander verbunden sind. Über diese Brücke kommunizieren 1 bis 4 Rechnerpaare miteinander (siehe Abbildung 3.1.1). Die Brücke arbeitet wahlweise im Store & Forward- oder CutThrough-Modus. Betrachten wir zuerst die Kommunikation im CutThrough-Modus.

Hierzu wurden wiederum Datenübertragungen mit unterschiedlichen Paketgrößen simuliert. Die maximal übertragbare Rahmengröße wurde auf 4200 Byte festgelegt. Die Abbildungen 3.3.1a und 3.3.1b zeigen einen Ausschnitt der Speicherbelastung der Bridge über der Zeitachse während einer Datenübertragung zwischen zwei Endgeräten (Rechner) bei Verwendung einer Paketgröße von 32 Byte und einer Fenstergröße von 2 Rahmen. Die Abbildung 3.3.1b zeigt einen Ausschnitt (Zeitindex 0 – 1000) der Simulation aus Abbildung 3.3.1a, um die Vorgänge im Speicher besser sichtbar zu machen.

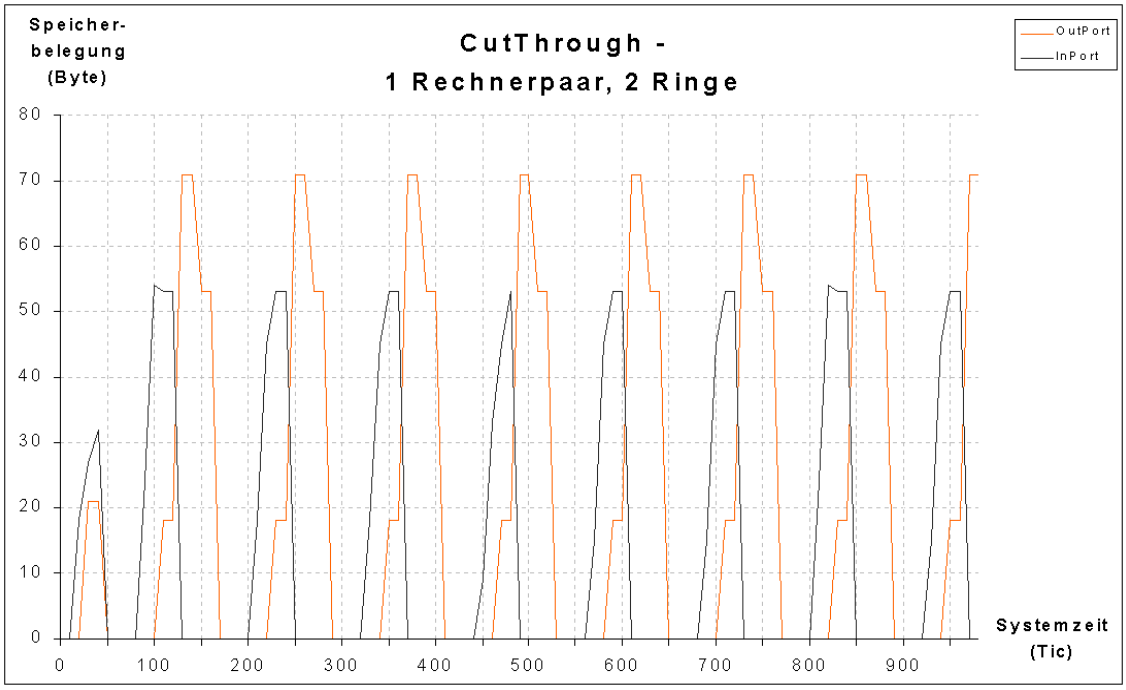


**Abbildung 3.3.1b:** Speicherbelastung (Simulation) der Ein- und Ausgangspuffer einer Brücke bei Kommunikation zwischen zwei Rechnern im CutThrough-Modus. Paketgröße – 32 Byte; Fenstergröße – 2 Rahmen.

Es ist erkennbar, daß die Belastungskurve des Ausgangspuffers (orange) gegenüber der des Eingangspuffers (schwarz) zeitversetzt ansteigt und fällt. Auffällig ist ebenfalls, daß der Puffer des Zielports stärker belastet wird als der des Quellports. Ein Grund hierfür ist, daß die Brücke auf ein freies Token im Zielring warten muß, bevor der am Ausgabeport anliegende Rahmen weitergeleitet werden kann. Da die durch die Brücke zur

Adreßauflösung benötigte Zeit (entspricht ca. der Zeit zum Empfang von 30-40 Byte) geringer ist als die Zeit die notwendig ist, um einen kompletten Datenrahmen (für 32 Byte-Paket) inklusive Header und Trailer zu empfangen, wird der Datenrahmen zum Ausgangsport weitergeleitet während er am Eingangsport noch empfangen wird.

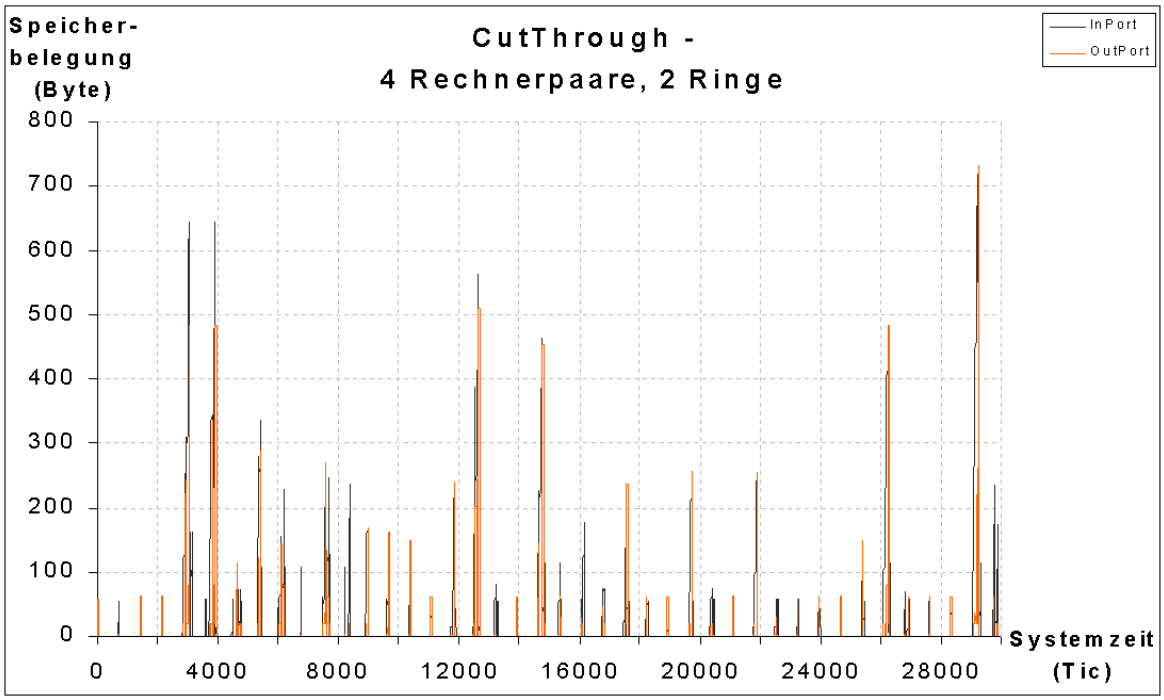


**Abbildung 3.3.2:** Speicherbelastung (Simulation) der Eingangs- und Ausgangsports einer Bridge die zwei Ringe miteinander verbindet und über die 4 Rechnerpaare miteinander kommunizieren. Die Paketgröße wurde für alle Paare auf 2 Kbyte festgelegt. Maximale Rahmengröße - 4200 Byte; Window – 2 (kein Einfluß); Übertragungsmodus – CutThrough.

Zusätzlich kommt zu der Speicherbelastung durch die Datenrahmen noch die der Kontrollrahmen hinzu. Nachdem der Bestätigungsrahmen für das komplett und korrekt übertragene Datenpaket empfangen wurde und der darauf folgende Rahmen mit der Sequenznummer des Datenpaketes zurückgesandt worden ist, kann sofort der nächste Datenrahmen an den Empfänger abgeschickt werden (erkennbar durch Ansteigen der Speicherbelegung beim Übertragen des zweiten Datenrahmens (Bereich zwischen 0 und 100 Tics in Abb. 3.3.1b).

Dieses Verhalten ist auch bei der Simulation von vier über eine Brücke miteinander kommunizierenden Rechnerpaaren zu beobachten (siehe Abbildung 3.3.3). Ebenfalls zu erkennen ist, daß die Speicherbelastung in diesem Fall stärker auf Seiten des Eingangspuffers liegt.

Das Maximum für die Belegung des Speichers am Eingangsport ist ca. viermal so hoch wie bei der Kommunikation zwischen nur zwei Rechnern. Tabelle 3.3.1 zeigt das nocheinmal in konkreten Zahlen.

| | 1 Rechnerpaar | | 4 Rechnerpaare | |
|---|---|---|---|---|
| | Maximum | Durchschnitt | Maximum | Durchschnitt |
| Eingangsport | 57 Byte | ≈ 11 Byte | 216 Byte | ≈ 170 Byte |
| Ausgangsport | 71 Byte | ≈ 24 Byte | 71 Byte | ≈ 24 Byte |

**Tabelle 3.3.1:** Speicherbelastungen der Eingangs- und Ausgangsports der Brücke bei unterschiedlichem Datenaufkommen, konstanter Paketgröße von 32 Byte und gleicher Performance aller beteiligten Rechner.

Somit besteht eine lineare Abhängigkeit zwischen der Anzahl der über die Brücke kommunizierenden Rechnerpaare, also der Auslastung der Übertragungskapazität des Ringes und der maximalen Speicherbelegung, bei gleichen Rahmenbedingungen wie Paketgröße und Paket- und Rahmengenerierungsleistung der Sender bzw. Empfänger. Die Belastung des Ausgangspuffers hingegen bleibt konstant. Anhand dieser Ergebnisse ist nachvollziehbar, daß bei einer hinreichend großen Anzahl von Rechnern im Netz und dem Auftreten eines Broadcast-Sturmes die Brücken unter Umständen nicht mehr in der Lage sind, die große Menge an kleinen Rahmen zu bewältigen, da die Kapazität der Puffer der Eingangsports nicht mehr ausreicht sie solange zwischenzuspeichern, bis durch die Adreßauflösung der Zielring (Ausgangsport) bestimmt wurde.

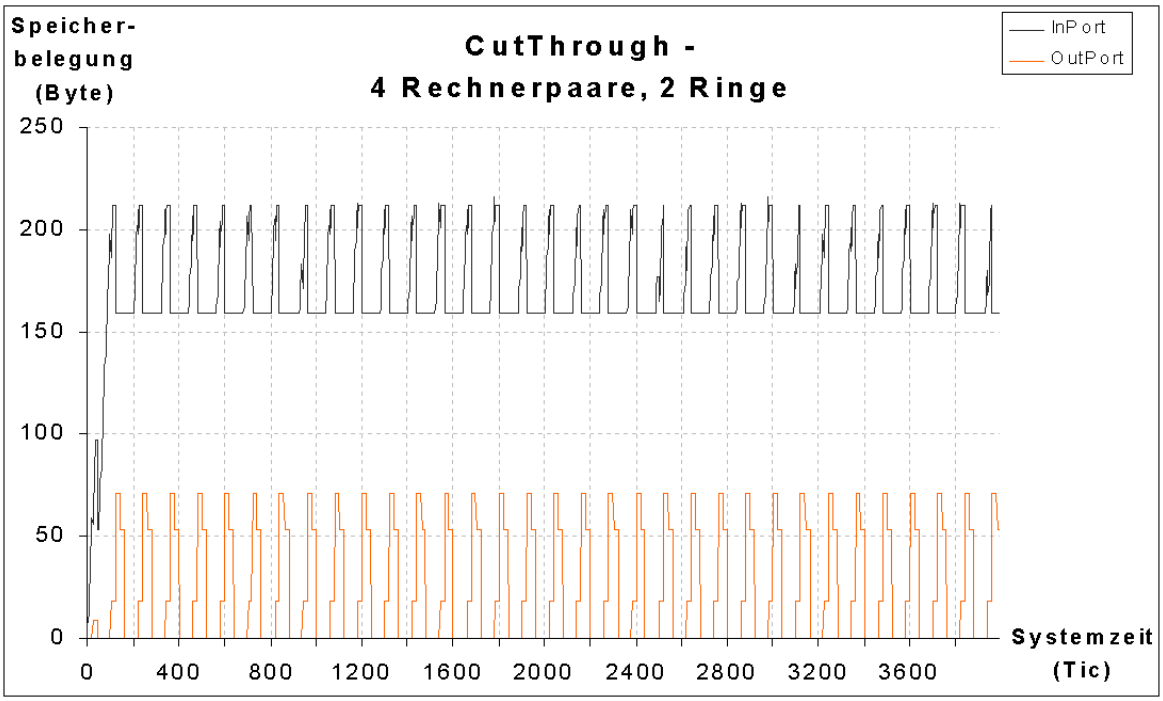


**Abbildung 3.3.3:** Speicherbelastung (Simulation) der Eingangs- und Ausgangsports einer Bridge die zwei Ringe miteinander verbindet und über die 4 Rechnerpaare miteinander kmmunizieren. Die Paketgröße wurde auf 32 Byte festgelegt. Weitere wichtige Parameter: MTU – 4200 Byte; Window – 2; Übertragungsmodus – CutThrough.

Verwendet man größere Datenpakete zur Kommunikation, so verliert die Zeitverzögerung durch die Adreßauflösung zunehmend an Einfluß auf die Auslastung des Speichers der Eingangsports und andere Einflüsse, wie die Wartezeit auf ein freies Token in Quell- und Zielring, werden bestimmend für die Charakteristik des Graphen für die Speicherbelegung über der Zeit (Abbildung 3.3.2). Als Folge der Verschiebung der Einflußsphären, gleichen sich die Belastungskurven für die Eingangs- und Ausgangsports einander an. So liegt das Maximum der gemessenen Werte für die Belegung des Eingangspuffers bei 726 Byte und der Durchschnitt über allen Werten bei ≈ 19 Byte. Das Maximum für den Ausgangspuffer beträgt 732 Byte und die durchschnittliche Belastung (≈ 9 Byte) ist etwas niedriger als die des Eingangspuffers. Grund hierfür ist der Einfluß der Verweilzeit der ankommenden Rahmen im Eingangspuffer durch die Adreßauflösung. Der etwas höhere Wert für das Maximum der Speicherbelastung ergibt sich durch die Wartezeit auf ein freies Token im Zielring. Da die Netzlasten auf Quell- und Zielring aber realtiv symetrisch ausfallen – alle Kommunikationspaare besitzen die selben Quell- und Zielringe – wirken sich die Wartezeiten auf ein freies Token im Zielring nicht in dem selben Maße auf die durchschnittliche Speicherbelastung aus.

Arbeitet die Bridge im „Store and Forward“-Modus, so werden Ein- und Ausgangspuffer gleichermaßen stark belastet (Abbildung 3.3.4), da die Rahmen in jedem Fall erst komplett am Eingangsport empfangen werden müssen, bevor sie an den Ausgangsport weitergeleitet werden können. Die zeitliche Verzögerung durch die Adreßauflösung spielt somit nur bei sehr kleinen Rahmen eine Rolle. In Abb. 3.3.4 ist zu sehen, wie die einzelnen Rahmen am Eingangsport (*InPort*; schwarze Linie) sukzessive eingelesen, an den Ausgabeport (*OutPort*; orange Linie) übergeben und bei Erhalt eines freien Tokens in den Zielring weitergegeben werden.

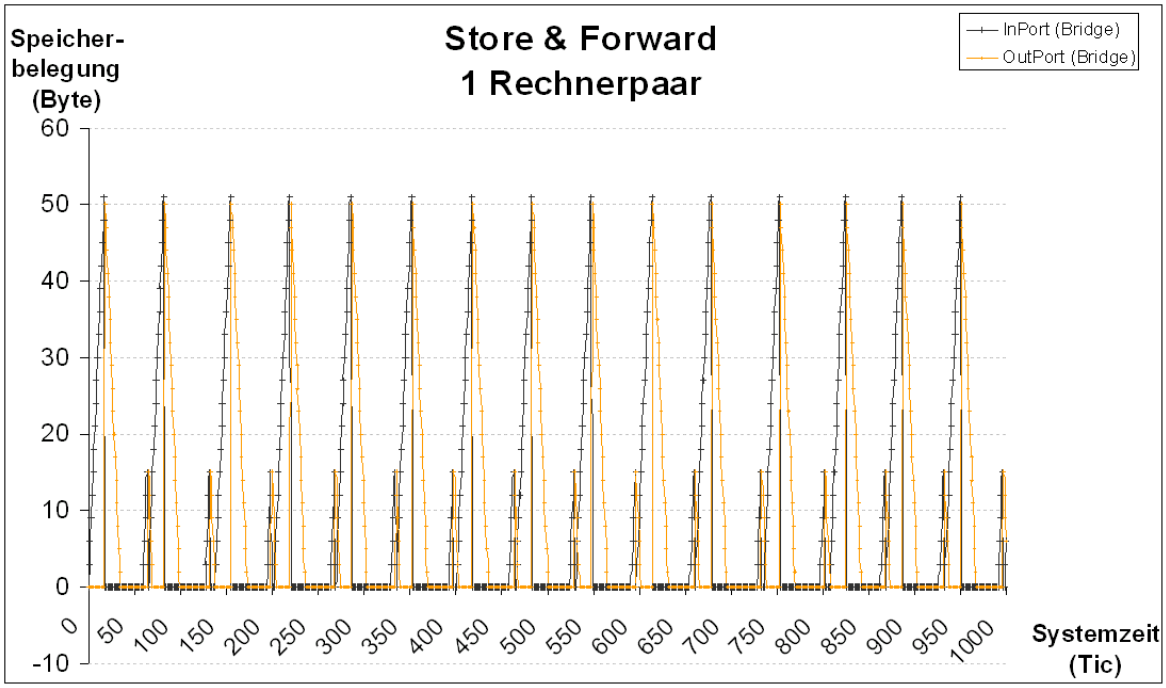


**Abbildung 3.3.4:** Speicherbelastung (Auszug) einer Bridge/Switch über der Zeit bei einer Datenübertragung im Store & Forward-Modus (Simulation) ohne zusätzliche Netzlast; Paketgröße 32 Byte; MFS/MTU=4200 Byte

Wird zusätzlich Netzlast erzeugt, durch Hinzunahme von untereinander kommunizierenden Lastrechnerpaaren, so zeigt sich, daß die Speicherbelastung hierdurch nur geringfügig beeinflußt wird. Die Abbildung 3.3.5 zeigt die Speicherbelastung bei Hinzunahme von 3 Lastrechnerpaaren. Grund für die geringe Auswirkung auf die Speicherbelastung ist unter anderem die gleichmäßige Verteilung der Netzlast auf beide Ringe, da alle simulierten Kommunikationen die selben Quell- und Zielringe besitzen. Wird ein Ring stärker belastet, so steigt die Speicherbelastung für den zu diesem Ring gehörenden Port entsprechend, da sich die Wartezeiten auf ein freies Token in dem entsprechenden Ring verlängern. Da das Simulationsmodell keine Prioritätssteuerung beinhaltet, liegen die so erhaltenen Werte für die Speicherbelastung über denen für ein äquivalentes reales System.

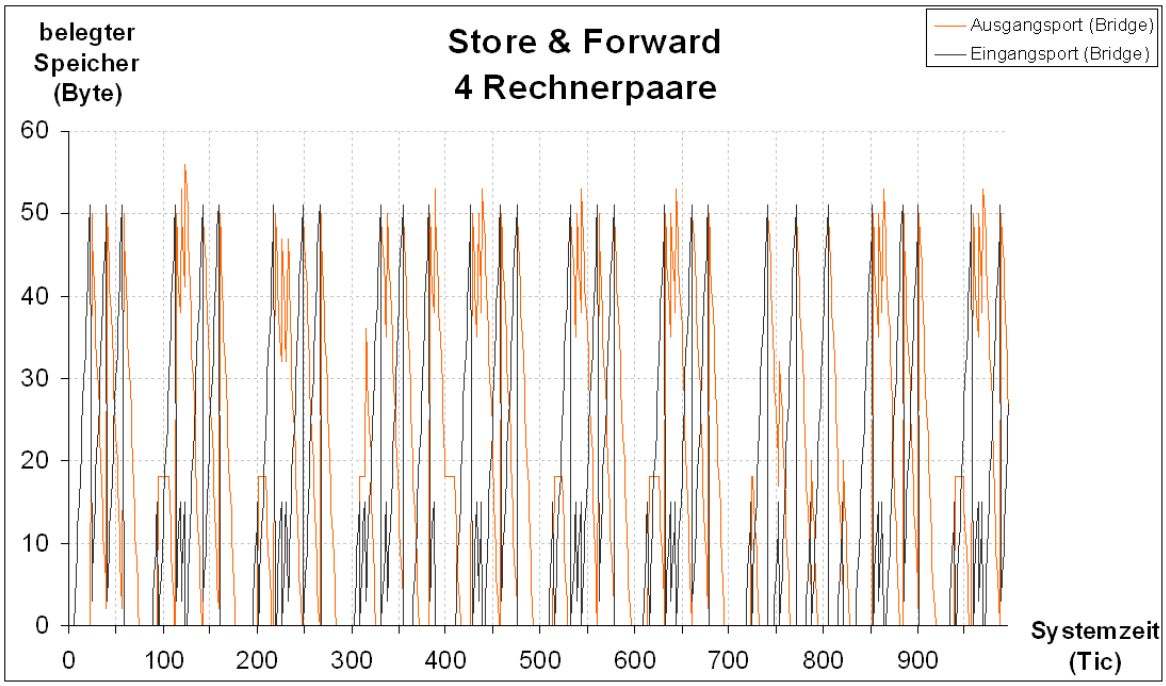


**Abbildung 3.3.5:** Speicherbelastung (Auszug) der Ein- und Ausgangsports einer Bridge/Switch über der Zeit bei einer Datenübertragung im Store & Forward-Modus (Simulation) mit zusätzlicher Netzlast (4 Lastrechnerpaare); Paketgröße 32 Byte; MFS/MTU=4200 Byte

In Abb. 3.3.5 sind einige Spitzen in der Speicherbelastung des Ausgangsports (z.B. bei Zeitindex 120 und 320) im Vergleich zur Speicherbelastung des Eingangsports erkennbar. Dieses Verhalten resultiert aus Wartezeiten auf ein freies Token im Zielring. Demgegenüber würden sich Wartezeiten auf ein freies Token im Quellring durch geringere Belastungen des Speichers des Eingangsports (ähnliches Verhalten, wie z.B. bei Zeitindex 700) bemerkbar machen. Ein solches Phänomen tritt in diesem Simulationsbeispiel aber nicht auf, da die gesamte Kommunikation, die über die beiden Ringe läuft, auch über die Ein- und Ausgangsports der Bridge geht.

Verändert man die Größe der zu sendenden Datenrahmen, so hat dies Einfluß auf die Speicherbelastung der Bridge, ähnlich wie bei Verwendung des CutThrough-Modus. Die Abbildung 3.3.6 zeigt die Speicherbelastung der Ein- und Ausgangsports einer Bridge (Store & Forward–Modus) bei Verwendung von 4Kbyte großen Rahmen zur Datenübertragung. Es ist zu erkennen, daß jeweils immer eines der kommunizerenden Rechnerpaare darauf warten muß, daß der entsprechende Ring frei ist, um seine Datenrahmen an den jeweiligen Empfänger absetzen zu können (zwei aufeinander folgende Spitzen; schwarzer Graph; Zeitindex 0 - 3000).

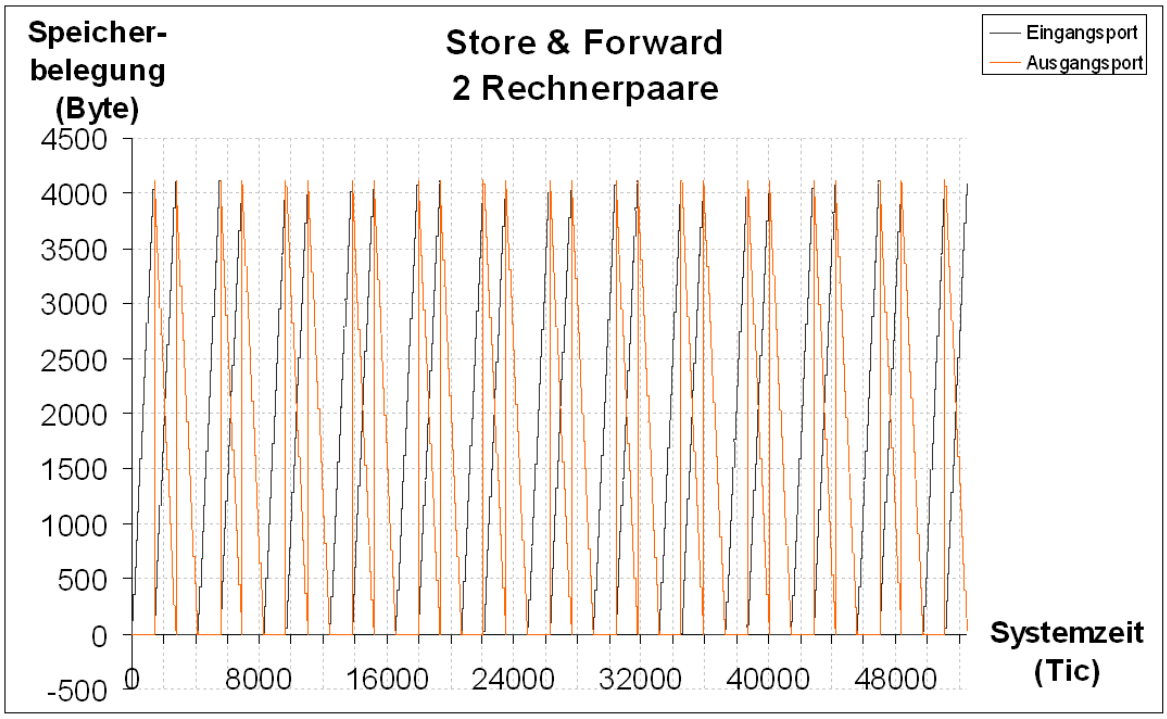


**Abbildung 3.3.6:** Speicherbelastung (Simulation) von Ein- und Ausgangsport einer Bridge (Store & Forward – Modus) über der Zeitachse bei Verwendung von Datenrahmen der Größe 4Kbyte. Einsatz eines zusätzlichen Lastrechnerpaares.

Hieraus und aus der Tatsache, daß kein weiterer Datenverkehr als der aus den beiden parallel laufenden Kommunikationen (1 Meßrechner- und 1 Lastrechnerpaar nach Abb. 3.3.1) existiert, resultiert die in Abb. 3.3.6 sichtbare Periodizität mit einem Intervall von durchschnittlich 4147,1 Tics. Die geringfügigen Schwankungen der Spitzenbelastungswerte sind wiederum Folge der variablen Verarbeitungszeiten der beteiligten Rechner und der Bridge bzw. der Wartezeiten auf ein freies Token im Zielring. Nachfolgend eintreffende Rahmen erhöhen somit die Speicherbelastung.

Die Tabelle 3.3.2 verdeutlicht nocheinmal die Abhängigkeiten zwischen Paketgröße, Netzlast und Speicherbelastung der Bridge. Deutlich erkennbar ist die Auswirkung der veränderten Rahmen-/Paketgröße: Je größer die übertragenen Rahmen, desto größer auch die Speicherbelastung.

Es zeichnet sich ebenfalls die Tendenz ab, je mehr Datenverbindungen unter Verwendung von kleinen Rahmen über die selben Ports laufen, desto größer die Speicherbelastung dieser Ports. Dieses Verhalten in der Simulation bestätigt somit das aus realen Systemen bekannte Phänomen der Überlastung der Bridge infolge eines auftretenden Beaconings bzw. Broadcast-Stürmen (bei Wiederanlaufen eines Netzes nach Teil- bzw. Komplettausfall).

| | 32 Byte Paketgröße | | 4096 Byte Paketgröße | |
|---|---|---|---|---|
| | Speicherlast Maximum | Speicherlast Durchschnitt | Speicherlast Maximum | Speicherlast Durchschnitt |
| 1 Rechnerpaar | | | | |
| Eingangsport | 51 Byte | ≈ 8 Byte | 4116 Byte | ≈ 1018 Byte |
| Ausgangsport | 50 Byte | ≈ 7 Byte | 4114 Byte | ≈ 1017 Byte |
| 2 Rechnerpaare | | | | |
| Eingangsport | 51 Byte | ≈ 12 Byte | 4116 Byte | ≈ 1399 Byte |
| Ausgangsport | 53 Byte | ≈ 15 Byte | 4117 Byte | ≈ 1349 Byte |
| 4 Rechnerpaare | | | | |
| Eingangsport | 51 Byte | ≈ 14 Byte | 4116 Byte | ≈ 1687 Byte |
| Ausgangsport | 56 Byte | ≈ 19 Byte | 4117 Byte | ≈ 1651 Byte |

**Tabelle 3.3.2:** Speicherbelegung für Ein- und Ausgangsport einer Brücke bei veränderten Paketgrößen und unterschiedlicher Netzlast, durch Variation der Anzahl der gleichzeitig laufenden Kommunikationen. Übertragungsmodus – Store and Forward; maximale Rahmengröße – 4200 Byte.

Aus der Tabelle 3.3.2 geht aber auch hervor, daß mehrere konkurrierende Verbindungen über diese Bridge notwendig sind, um die Speicherkapazität einer Bridge zu überlasten.

# 4 Auswertung und Schlußfolgerungen

Das in dieser Arbeit erarbeitete Simulationsmodell ist besonders für Untersuchungen bezüglich der kommunikativen (protokolltechnischen) Abläufe von Datenverbindungen und zur Betrachtung von internen Abläufen der Netzkopplungsgeräte geeignet (siehe Abschnitt 3). Aufgrund der verschiedenen Möglichkeiten einer genauen Protokollierung der einzelnen Systemzustände, können die Abläufe in dem simulierten System gut nachvollzogen bzw. analysiert und etwaige Abhängigkeiten bestimmt werden.
Die Ergebnisse aus Abschnitt 3 zeigen, daß das gewählte Simulationsmodell in der Lage ist, Datenübertragungen über ein TokenRing-Netzwerk sehr realistisch zu simulieren. Es können übertragungsspezifische (z.B. Datendurchsatz, Übertragungszeit) als auch gerätespezifische Daten (z.B. max./durchschnittliche Speicherbelastung) per Simulation ermittelt werden.

Mittels der im Verlauf der Arbeit durchgeführten Simulationen konnte gezeigt werden, daß die Leistungsfähigkeit eines Netzes nicht nur abhängig ist von der Leistungsfähigkeit der verwendeten Technik, sondern in hohem Maße auch von den zur Übertragung verwendeten Protokollen. Es konnte nachgewiesen werden, daß sich durch die Veränderung der Parameter für die verwendeten Protokolle die Übertragungseigenschaften einer Verbindung und somit ihre Leistungsfähigkeit (Datendurchsatz; Abschnitt 3.2) ändert. Auch konnte gezeigt werden, daß sich die Wahl dieser Parameter auf die Anforderungen an die Hardware (Speicherbelastung; Abschnitt 3.3) auswirkt. Es hat sich aber auch bestätigt, daß mit steigender Anzahl von konkurrierenden Datenverbindungen auf ein und demselben Ring(-system) die Performance desselben sinkt. Auf diese Weise werden für eine einzelne Datenverbindung ohne zusätzliche Netzlast herausgearbeitete Performance-Gewinne wieder relativiert. Daraus läßt sich ableiten, daß die Parameter für eine Datenverbindung günstigerweise immer in Abhängigkeit von der aktuellen bzw. zu erwartenden Netzbelastung und Leistungsfähigkeit der Kommunikationspartner gewählt werden sollten. Entsprechende Mechanismen sind ansatzweise in verschiedene Protokolle (TokenRing: Rahmenfensterprotokoll; TCP/IP: Rahmengenerierung) integriert worden (vergl. [Tan92]).

Als wichtige Einflußfaktoren für eine Datenverbindung in Bezug auf die Performance konnten die folgenden Parameter herausgearbeitet werden:

- Wahl der Rahmengröße
- Paketgröße
- Rahmenfenstergröße
- Anzahl der Hops (Anzahl der zu durchlaufenden Brücken)
- Übertragungsmodus der Brücken (Store & Forward vs. CutThrough)
- Netzlast
- Serverlast.

Der Einfluß der TokenRing-Prioritätssteuerung konnte nicht ermittelt werden, da dieser Mechanismus keine Berücksichtigung im vorliegenden Modell fand.

Weiterhin zeigte sich, daß das vorliegende Simulationsmodell besonders dazu geeignet ist gering dimensionierte TokenRing-Netzwerke zu simulieren. Aufgrund der Modellstruktur, der Genauigkeit in der Darstellung der einzelnen Systemzustände und der Überführungsfunktionen in den Folgezustand, ist der Simulator im Gegensatz zu Simulatoren auf Basis mathematischer Modelle (siehe Abschnitt 1) nicht dazu geeignet größere Systeme darzustellen. Ein weiteres Problem, was einem derartigen Einsatz entgegensteht, ist die Tatsache, daß größere Netze meist eine heterogene Struktur aufweisen. Es werden in einem solchen Netz oft unterschiedliche Übertragungsprotokolle, Netzwerktypen und Teiltopologien eingesetzt. Da das vorliegende Simulationsmodell in dieser Hinsicht auf das NetBIOS-Protokoll in einem TokenRing-System beschränkt ist, können entsprechend davon abweichende Strukturen bzw. Eigenschaften nicht bzw. nicht hinreichend nachgebildet werden.

# 5 Ausblicke

In den vorhergehenden Abschnitten wurden Ausgangspunkte, Ziele, Nutzen und Realisierungsmöglichkeiten der Simulation von Netzwerken und im speziellen von TokenRing-Netzwerken erörtert. Weiterhin wurde ein konkretes Projekt zu deren Simulation vorgestellt und Ergebnisse diskutiert. Der folgende Abschnitt soll Möglichkeiten aufzeigen und Denkanstöße zur Weiterentwicklung des beschriebenen Simulators geben.

MadQ wurde ursprünglich als ein Simulator mit alphanumerischer Ausgabe konzipiert. Während der Realisierung des Modells erwies sich eine zumindest rudimentäre Grafikfähigkeit zur Darstellung systeminterner Prozesse (Zustandsänderungen der einzelnen Warteschlangen u. ä.) als sinnvoll und notwendig, da die Verwendung einer grafischen Ausgabe nicht nur Rückschlüsse auf die Korrektheit des Systems und der eingebetteten Abläufe zuläßt, sondern auch eine günstige Repräsentation von internen Zuständen, Zustandsänderungen und Meßwerten des Systems darstellt. Es ist vorstellbar und wünschenswert, die Grafikfähigkeit des Simulators weiter zu vervollkommnen bzw.eine komplette Suit mit Präsentations- und Analysefunktionalität zu schaffen. Möglich wäre auch die Zusammenfassung aller Funktionen in einem einzigen Programm. Somit hätte der Nutzer zu jeder Zeit den Überblick über die Ausgangsparameter, Zustandsänderungen des Systems und wäre in der Lage, Ergebnisse sofort abzulesen und in den Kontext der Messungen einzuordnen. Dem Nutzer wird es hiermit möglich, zu einem frühest möglichen Zeitpunkt erste Werturteile über das simulierte System abgeben zu können.

Ein einfaches Tool zur grafischen Darstellung der gemessenen Werte unter DOS wurde bereits erstellt. Denkbar und wünschenswert ist eine Portierung des Simulators und der Präsentations- und Analysetools auf ein 32-Bit Betriebssystem mit grafischer Bedieneroberfläche, wie Windows 95/NT, womit zum einen ein erheblicher Performance-Gewinn und zum anderen eine bessere Handhabbarkeit und Visualisierungsmöglichkeit verbunden wäre. Die Rechenleistung des Simulators kann unter einem solchen Betriebssystem ebenfalls gesteigert werden, indem Multithreading eingesetzt wird. Unter Multithreading versteht man die Aufteilung von monolitischen Prozessen in verschiedene

Teilprozesse, die alle auf dem selben Adreßraum arbeiten, nämlich dem des Prozesses dem sie alle angehören [Dei90]. Der Vorteil der Nutzung des selben Adreßraumes und die Zugehöigkeit zu jeweils dem selben Prozeß besteht darin, daß bei einem Task switch (dem Umschalten zwischen zwei Aufgaben) innerhalb eines Prozesses, nicht die gesamte Prozeßumgebung der letzten Task zu speichern und die der neuen Task zu restaurieren. So kann gegenüber herkömmlicher Parallelisierung Rechenzeit eingespart werden.

Die Zuweisung von CPU-Zeit erfolgt, wie auch bei herkömmlichen Prozessen durch eine Prioritätssteuerung. Der Thread, der die höchste Priorität besitzt und sich im READY-Status befindet wird zuerst ausgeführt. Betriebssysteme wie OS/2, Windows 95 oder Windows NT unterstützen die multiple Thread-Ausführung.

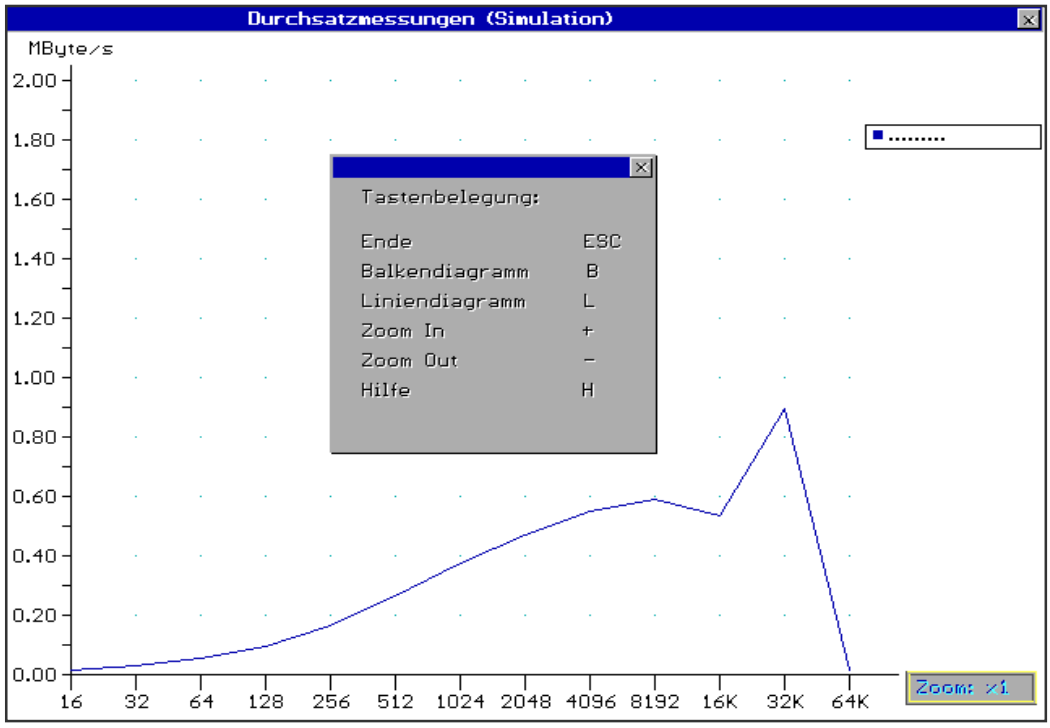


**Abbildung 5.1:** Bildschirmkopie des Präsentationstools ShowDiag für DOS mit geöffnetem Hilfefenster. Dient der grafischen Darstellung der Meßergebnisse am Simulator MadQ. Es können Linien und Balkengrafiken dargestellt werden. Als Eingabedateien dienen durch die Ausgabefunktion des Simulators bereitgestellte ASCII-Dateien.

Da verschiedene Betriebssysteme in Abhängigkeit von der genutzter Hardware in der Lage sind die einzelnen Threads auf verschiedene Prozessoren (so vorhanden) zu verteilen, bildet die Verwendung von Multithreading den ersten Schritt in Richtung einer Parallelisierung der Simulation [Luk93] und etwaiger zugehöriger Tools. Erste Ansätze bezüglich der Portierung des Präsentationstools sind schon vorhanden, wurden aber nicht in den Umfang dieser Arbeit miteinbezogen.

Eine weitere Möglichkeit der Weiterführung des Projektes bietet sich in der Erweiterung des Simulationsmodells durch Einbeziehung anderer gebräuchlicher Protokollimplementierungen und Übertragungstechniken wie TCP/IP (*transfer control protocol* / *internet protocol*) oder ATM (*asynchronous transfer mode*).

Ebenfalls sinnvoll ist eine weitere Verfeinerung des vorliegenden Simulationsmodells (NetBIOS) durch die Einbeziehung der höheren Protokollschichten. Dies kann zu weiteren Erkenntnissen bezüglich der Optimierung des Kommunikationsverhaltens verschiedener Anwendungen und Anwendungsgruppen führen, besonders in Hinblick auf die wachsende Bedeutung der Übertragung multimedialer Inhalte über das Netz.

Ein weiterer Punkt, welcher eine genauere Vorhersage und Analyse des Systemverhaltens eines TokenRing-Netzwerkes erlauben würde, wäre die Aufnahme einer Prioritätssteuerung für den TokenRing in das Modell.

Ziel der Bemühungen sollte es sein, ein umfassendes Werkzeug zur Simulation und Analyse von Netzwerken auf Basis einer grafischen Benutzerschnittstelle zu schaffen, so daß ein zu simulierendes System beispielsweise via „Drag and Drop“ visuell modelliert getestet und analysiert werden kann. Auf diese Weise wird das Modellieren transparent und überschaubar für den Nutzer und Fehler bei der Eingabe von systembeschreibenden Initialdaten werden so minimiert bzw. ganz ausgeschlossen.

# Literaturverzeichnis


[Bolxx] Bolch: ‚Leistungsbewertung von Rechnersystemen mittels Analytischer Warteschlangenmodelle'; Teubner Verlag Stuttgart

[Bha95] S. Bhattacharyya, J.T. Buck, W.-T. Chang, M.J. Chen, B.L. Evans, E.E. Goei, Ha, P. Haskell u. a.: ‚The Almagest' Vol. 1 - 4; Universtity Of California, Dep. of Electr. Eng. and Comp. Sci.; Berkeley (1990 - 95)

[Cor95] D. Corbett: ‚Cut-through Token Ring Switching & The Madge Smart Ringswitch'; Madge Networks Ltd. (July 1995)

[Dei90] H. M. Deitel: ‚An Introduction to OPERATING SYSTEMS' (Second Edition); Addison-Wesley (1990)

[Dem89] J. Demel: ‚JANAP - Ein Programm zur Simulation von elektrischen Netzwerken'; Dissertation, TU Wien, Fakultät f. Elektrotechnik (1989)

[Els88] J. Elsing: ‚MSDOS-Assembler-Programmierung (Praktische Anwendungen von DOS-Aufrufen in Maschinensprache)'; IWT Verlag München (1988)

[Gaw95] S.C. Gawne (Madge Networks Ltd.): ‚Cut-Through Token Ring Switching - A Detailed Study of Cut-Through Switching on Token Ring Networks'; Madge Networks Limited (1995)

[Gel67] W. Gellert, Dr. H. Küstner, Dr. M. Hellwich, H. Kästner: ‚Kleine Enzyklopädie Mathematik'; VEB Verlag Enzyklopädie Leipzig (1967)

[Has95] H. Hassenmüller: ‚Schneller Ring - Token Ring Switch mit ATM-Schnittstelle'; GATEWAY (Jan./Febr. 1995)

[Hod97] J.P.E. Hodgson: ‚Networks and Distributed Systems'; Saint Joseph's University Philadelphia (1997)

[Kau94] F.-J. Kauffels: ‚Moderne Datenkommunikation (eine strukturierte Einführung)'; DATACOM Verlag Bergheim (1994)

[Ker95] H. Kerner: ‚Rechnernetze nach OSI' 3. Auflage, Addison - Wesley Bonn (1995)

[Kiy91] K. Kiyek, F. Schwarz: ‚Mathematik für Informatiker 2'; Teubner Verlag Stuttgart (1991)

[Kre94] F. Kresse: ‚Token Ring-Netzwerke: Funktionsweise, Planung, Trouble-shooting‘; Hüthig Buch Verlag Heidelberg (1994)

[Lan89] J. Lange: ‚Turbo Pascal Version 5.5‘; Hüthig Buch Verlag Heidelberg (1989)

[Luk93] P. Luksch: ‚Parallelisierung ereignisgetriebener Simulationsverfahren auf Mehrprozessorsystemen mit verteiltem Speicher‘; Verlag Dr. Kovac´s Hamburg (1993)

[Mat90] R. Mathar, D. Pfeifer: ‚Stochastik für Informatiker‘; Teubner Verlag Stuttgart (1990)

[Mdg95] ‚Network Performance and the Client Connection‘; Madge Networks Ltd. (1995)

[Mes95] D. Messina, S.C. Gawne: ‚Token Ring Switch Evaluation Guide‘; Madge Networks Ltd. November (1995)

[Pfl86] G.Ch. Pflug: ‚Stochastische Modelle in der Informatik‘; Teubner Verlag Stuttgart (1986)

[Rei90] F. Reinhardt, H. Soeder: ‚dtv-Atlas zur Mathematik - Tafeln und Texte‘ Band 1 und 2, 8. Auflage; dtv München (1990)

[RFC02] IEEE 802.5 Reference Paper – Token Ring; RFC 1002 (1985)

[Sch97] T.R. Schmidt: ‚Performance und Einsatzmöglichkeiten von Token-Ring-Switches‘; Institut für Informatik Universität Leipzig (1997)

[Tan92] A. S. Tanenbaum: ‚Computer-Netzwerke‘ 2. Aufl.; Wolfram‘s Fachverlag Attenkirchen (1992)

[Tay94] M. Taylor (Madge Networks Ltd.): ‚Token Ring Switching - A Technology White Paper‘; Madge Networks Limited (1994)

[Vas96] N. Drakos: ‚Performance Modeling and Analysis of Computer Communication Networks‘ (transl./ed.: K. Vastola: http://pisson.ecse.rpi.edu/~vastola/pslinks/perf/node01.html, 1996); ComputerBased Learning Unit, University of Leeds (1993 - 1994)

[Wol96] J. Woller: ‚The Basics of Monte Carlo Simulations‘; University of Nebraska-Lincoln (1996)

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Symbolverzeichnis

| | |
|---|---|
| $\mathbb{N}$ | natürliche Zahlen |
| $\mathbb{N}_0$ | natürlichen Zahlen vereinigt mit 0, $\mathbb{N} \cup \{0\}$ |
| $N$ | Anzahl der Kunden im (Warteschlangen-) System in laufendem Betrieb |
| $N_q$ | Anzahl der Kunden in der Warteschlange in laufendem Betrieb |
| $N_s$ | Anzahl der Kunden die bedient werden in laufendem Betrieb |
| $q$ | Zeit die ein Kunde in der Warteschlange verbringt |
| $s$ | Zeit die ein Kunde in der Verarbeitung verbringt |
| $w$ | Zeit die ein Kunde im (Warteschlangen-) System verbringt, $w=q+s$ |
| $\tau$ | Interarrival time - Zeit zwischen zwei erfolgreichen Ankünften von Kunden für die Verarbeitung |
| $\lambda$ | durchschnittliche Ankunftsrate für Kunden des Warteschlangensystems, $\lambda = 1/E(\tau)$ |
| $\lambda_T$ | durchschnittlicher Durchsatz eines Computersystems (Jobs/Zeiteinheit) |
| $E(s)$ | Erwartete Servicezeit für einen Kunden |
| $E(\tau)$ | Erwartete interarrival time |
| $\mu$ | durchschnittliche Servicerate für einen Server $\mu = 1/E(s)$ |
| $\rho$ | Serverauslastung |
| $p_{id}$ | Wahrscheinlichkeit mit der ein Paket das Netz am aktuellen Knoten verläßt |
| $p_{ij}$ | Wahrscheinlichkeit der Wahl der Route von $Q_i$ nach $Q_j$ |

# Abkürzungen

| | |
|---|---|
| ACK | Bestätigung (engl.: *Acknowledgement*) |
| ATM | Asynchroner Übertragungsmodus (engl.: *asynchronous transfer mode*) |
| CM | “Durch Programm festgelegter Ablauf”- Modus (engl.: *compiled mode*) |
| CPU | Zentrale Verarbeitungseinheit (engl.: *central processing unit*) |
| CT | “Durchschieben” – Verarbeitungsmodus (engl.: *Cut Through*) |
| DM | *Disconnect Mode* |
| DOS | Plattenbasiertes Verarbeitungssystem (engl.: *disk operating system*) |
| DSAP | *Destination Service Access Point* |
| EI | Einheitsinkrement |
| FCFS | Kommt zuerst, wird zuerst bedient – Verarbeitungsregel (engl.: *first come first served*) |
| FIFO | Zuerst rein, zuerst raus – Verarbeitungsregel (engl.: *first in first out*) |
| IEEE | *Institute of Electrical and Electronics Engineers* |
| ISO | *International Standards Organization* |
| LAN | *Local Area Network* |
| LLC | Logische Verbindungsschicht (engl.: *logical link layer*) |
| MFS | Maximale Rahmengröße (engl.: *maximum frame size*) |
| MTU | Größte Übertragbare Einheit (engl.: *maximal transferable unit*) |
| PSN | Paketgeschaltetes Netzwerk (engl.: *packet switched network*) |
| RFC | Referenz (engl.: *reference*) |
| RR | Empfänger bereit (engl.: *Receiver Ready*) |
| SABME | *Set Asynchronous Balanced Mode Extended* |
| S&F | “Speichern und Weiterleiten” – Verarbeitungsmodus (engl.: *Store & Forward*) |
| SSAP | *Source Service Access Point* |
| TCP/IP | Übertragungskontrollprotokoll/ Internetprotokoll (engl.: *transfer control protocol / internet protocol*) |
| Tic | Systemzeitzähler – Einheitsbezeichner für die Zeit innerhalb eines Simulationssystems (engl.: *time incrementation count[-er]*) |

| | |
|---|---|
| TR | Token Ring – Bezeichnung für einen speziellen Netzwerktyp in Ringform |
| UA | *Unnumbered Acknowledgement* |
| WS | Arbeitsstation (engl.: *Workstation*) |
| XID | *Exchange Identification* |
| ZS | Zentrale Systemzeit(-fortschaltung) |

# Anhang

## A.1 Quelltext (MadQ.pas)

```
(*************************************************************************)
(*                                                                       *)
(* Program  : MadQ für DOS (Token-Ring-Netzwerk-Simulation)              *)
(* Version  : 0.92                                                       *)
(*                                                                       *)
(*************************************************************************)

program MadQ;                                          { Programmkopf       }

uses crt,dos,cursor,fileacc;                           { Einzubindg.d.Units }
                                                       { für I/O und File-  }
                                                       { zugriff.           }
const
    MaxLength   =50;
    MaxQueues   =5;                                    { Anz. Queues in     }
                                                       { einem Gerät.       }
    FHeaderSize =15;                                   { Größe des Rahmen-  }
                                                       { kopfes.            }
    FTrailerSize=6;                                    { Größe des Rahmen-  }
                                                       { Trailers.          }
    leer        =0;                                    { Warteschlangen-    }
    normal      =1;                                    { Statuskostanten.   }
    full        =2;
    LLC         =18;                                   { Rahmentypenkonst.  }
    None        =0;                                    {(dienen der Unter-  }
    DataFirstMid=68;                                   { scheidung in ihrer }
    DataOnlyLast=67;                                   { Verarbeitung.      }
    ACK         =32;
    bkcolor     =0;                                    { Hintergrundfarbe   }
                                                       { für die grafische  }
                                                       { Darstellung.       }
type
    QElPtr   = ^QElType;                               { Zeiger auf Warte-  }
                                                       { schlangenelement-  }
                                                       { eintrag.           }

    Elmnt    = record                                  { Warteschlangenelem.}
                 FT  :byte;                            { Rahmentypfeld      }
                 DA  :byte;                            { Zieladressfeld     }
                 SA  :byte;                            { Quelladressfeld    }
                 FS  :Word;                            { Rahmengrößenfeld   }
                 CntrlBlck:word;                       { Controllfeld       }
               end;
    QElType  = record                                  { Warteschlangen-    }
                 Element:Elmnt;                        { elementeintrag     }
                 n      :QElPtr;                       { + Zeiger auf das   }
               end;                                    { nächste Element.   }

    QType    = record                                  { Warteschlange (Q)  }
                 Name      :string;                    { Name (optional)    }
                 Ring      :byte;                      { Ringzugehörigkeit  }
                 Idx       :word;                      { Anz. Rahmen in Q   }
                 Size      :longint;                   { akt. Größe (Byte)  }
                 MaxQSize  :longint;                   { max. Größe (Byte)  }
                 Possblty  :real;                      { Wahrscheinl.-wert  }
                 RSeed     :longint;                   { Speicher für Zu-   }
                                                       { fallszahl          }
                 Lost      :word;                      { Verlorene Rahmen   }
```

```
                FirstElPtr:QElPtr;              { Zeiger auf 1. Ele. }
                LastElPtr :QElPtr;              { Zeiger auf letztes }
                                                { Element der Queue  }
                LastFType :byte;                { Typ des zuletzt    }
                                                { übertrag. Rahmens  }
                CalcFrame :boolean;             { Zeigt an ob gerade }
                                                { ein Rahmen bearb-  }
              end;                              { eitet wird.        }

    QPtr     = ^QSetType;                       { Zeiger auf eine    }
                                                { Menge von Wartesch.}

    QSetType = record                           { Liste von Wartesch.}
                Queue:QType;
                n    :QPtr;                     { Zeiger auf nächste }
              end;
    NetUnit  = record
                Name      :string;              { Name des Gerätes   }
                UnitType  :byte;                { Typ des Gerätes    }
                ADDR      :byte;                { QuellAdresse (SA)  }
                Dest      :byte;                { ZielAdrese (DA)    }
                QCount    :byte;
                WINDOW    :byte;                { WINDOW-Größe       }
                WantSeqLLC:byte;                { Ausstehnde Seq-LLCs}
                Rate      ,                     { Datendurchsatz     }
                GenPos    :real;                { Datengenerierungs- }
                                                { wahrscheinlichkeit }
                GenRSeed  :longint;             { Speicher für Zu-   }
                ACKRSeed  :longint;             { fallszahlen        }
                LeftData  :longint;
                LeftBytes :longint;             { Noch zu übertragen-}
                                                { de Bytes.          }
                XmitedBytes:longint;            { Übertragene Bytes  }
                XBs       :longint;             { dito d. akt. Pakets}
                first     :longint;             { Anz. d. ges. Frames}
                RecvdFrms :longint;             { Anz. d. empf.Frames}
                QSet      :QPtr;                { Warteschl. d.Geräts}
                GOTACK    :boolean;             { ACK empf.? JA/NEIN }
              end;
  LastrechnerTyp =array[1..15] of NetUnit;
  QArrayType=array[1..15] of QType;
var
    RingAnzahl      :byte;                          { Anzahl der Ringe
}
                                                { des Systems.       }
    MAX_THROUGH_PUT :word;                          { max.
Datendurchsatz}
                                                { (z.B.:4-16 Mbit/s) }
    LOGGING         :boolean;                   { Soll Verkehr mit-  }
                                                { gelogged werden ?  }
    PAKET_SEND      :boolean;
    LastRechner     :byte;                          { Anzahl Lastrechner
}
    TokenIsFree     :array[1..256] of boolean;  { Ist Token auf      }
                                                { Ring[I] frei?      }
    FixedSeed       :byte;                          { Fester Startwert
}
                                                { für Zufallsgener.? }
    QTop            :pointer;                   { Zeiger auf den     }
                                                { Speicheranfang.    }
    tmp             :QElPtr;                    { temporärer Zeiger  }
                                                { auf ein Element    }
                                                { einer Queue.       }
    INIFile         :string;                    { Name der INI-Datei }
    ASCIIFile       :string;                    { Name der Meßwert-  }
                                                { datei.             }
    OutFile         :string;
    LogFile         :string;                    { Name des Logfiles  }
    Qb0,Qb1,Qb2,Qb3 ,                               { Bez. von Warte-
}
    Q0,Q1,Q2,Q3     :QType;                         { schlangen.
}
```

```
    Qlb0,Qlb1,Qlb2  ,
    Qlb3,Ql0,Ql1,Ql2,
    Ql3             :QArrayType;
    Last            :LastrechnerTyp;                    { Lastrechner
}
    Unit1           :NetUnit;                           { Quellrechner
}
    DataSize        :longint;                           { zu übertragende
}
    FrameNumb       ,                                   { Daten(-rahmen)
}
    XmitFrameCount  :longint;                           { übertragene
}
                                                        { Rahmen
}
    err,i           :word;
    size            :string;
    ZeitTakt        ,                                   { Systemzeit-Tic
}
    fr              :longint;
    TimeFactor      :real;                                    {
Umrechnungsfaktor  }
                                                        { von Sim.->
Realzeit}
    MFSize          ,
    MaxFrameSize    ,
    RestPaketSize   ,
    RestFrames      ,
    PaketSize       :word;                                    { Paketgröße
}
    TraceStep       :word;                                    { Zeitl.
Abstand     }
                                                        { zw. den Unterbr.
}
    ACKFrame        ,
    LLCFrame        ,
    El              :Elmnt;
    SHOWLENGTH      ,                                   { Soll die Sim.
}
                                                        { gezeigt werden?
}
    PRNSTAT         ,                                   { Muß Ergebnis
}
                                                        { gedruckt werden?
}
    UseINI          ,                                   { Soll INI-Datei
}
                                                        { benutzt werden?
}
    TRACE           :boolean;                           { Soll getracet
}
                                                        { werden?
}
    x               :integer;
    fl              :text;
    XmitMode        :byte;                                    { Übertragungs-
Modus }

(*-------------------------------------------------------*)
(* Diese Funktion wandelt ein Longint-Wert (Numb) in    *)
(* einen Real-Wert (RealVal) um.                        *)
(*-------------------------------------------------------*)

Function RealVal(Numb:longint):real;
begin
RealVal:=numb;
end;

(*-------------------------------------------------------*)
(* Diese Funktion prüft,ob ein Rahmen vollständig über- *)
(* tragen wurde und liefert das Ergebnis als Wahrheits- *)
(* wert zurück.                                         *)
```

```
(*-------------------------------------------------------*)

Function FrameIsComplete(E:elmnt):boolean;
begin
If (E.FS = E.CntrlBlck) then FrameIsComplete:=true
else FrameIsComplete:=false;
end;

(*-------------------------------------------------------*)
(* Diese Procedure legt ein Log file an und trägt dort  *)
(* die den „Monitor“ durchlaufenden Rahmen mit Sender,  *)
(* Empfänger, Größe, Typ und Zeitstempel ein.           *)
(*-------------------------------------------------------*)

Procedure WriteLog(E:elmnt);
var FrameType:string[16];
begin
  case E.FT of                                          { Ermitteln des Rah-
}
    0 :FrameType:=‘None          ,;                     { mentyps.
}
    18:FrameType:=‘LLC           ,;
    32:FrameType:=‘ACK           ,;
67:FrameType:=‘Data Only Last‘;
68:FrameType:=‘Data First Mid‘;
end;
  if not Exists(LogFile) then begin                     { Falls Datei nicht
}
    assign(fl,LogFile);                                 { existiert, Datei
}
    rewrite(fl);                                        { erzeugen.
}
    writeln(fl,ZeitTakt:7,‘ WS‘,E.SA,‘-> WS‘,E.DA,      { Zeit, Sender,
Empf.}
      , ,,FrameType,‘  ,,E.FS:6);                       { Rahmentyp u. –
größe}
    close(fl);                                          { schreiben.
}
  end                                                   { Datei schliessen.
}
else begin
    assign(fl,LogFile);                                 { Existiert Datei,
}
    append(fl);                                         { dann öffnen und
}
    writeln(fl,ZeitTakt:7,‘ WS‘,E.SA,‘-> WS‘,E.DA,      { Daten anhängen.
}
            , ,,FrameType,‘  ,,E.FS:6);
    close(fl);                                          { Datei schliessen.
}
  end;
end;

(*-------------------------------------------------------*)
(* Diese Funktion fügt ein Element in eine Queue ein.   *)
(* War diese Aktion erfolgreich so liefert die Fuktion  *)
(* den Wert TRUE zurück.                                *)
(*-------------------------------------------------------*)

Function InsEl(E:elmnt;var Q:QType):boolean;
var temp:QElPtr;
begin
  if ((maxavail<10)                                     { Ist nicht genug
}
     or ((Q.Size+E.FS) > Q.MaxQSize)) then begin        { Speicherplatz vor-
}
    InsEl:=false;                                       { handen oder Q voll
}
    exit;                                               { dann gib Fehler
}
  end;                                                  { zurück.
}
```

```
  new(temp);                                              { SONST: erzeuge neu-
}
  temp^.n:=NIL;                                           { es Element und
}
  temp^.Element:=E;                                       { füge es an Liste
an}
  if (Q.FirstElPtr=NIL) then begin
    Q.FirstElPtr:=temp;
    Q.LastElPtr:=temp;
  end
  else begin
    Q.LastElPtr^.n:=temp;
    Q.LastElPtr:=Q.LastElPtr^.n;
  end;
  inc(Q.Size,E.FS);                                       { aktualisiere Größe
}
                                                          { der Warteschlange
}
  inc(Q.Idx);                                                    { und Anz. Der
Rahmen}
InsEl:=true;
end;

(*-------------------------------------------------------*)
(* Diese Funktion löscht ein Element aus einer Queue.   *)
(* War diese Aktion erfolgreich so liefert die Fuktion  *)
(* den Wert TRUE zurück.                                *)
(*-------------------------------------------------------*)

Function RmEl(var E:elmnt;var Q:QType):boolean;
var temp:QElPtr;
begin
  if (q.FirstElPtr<>NIL) then begin                       { Ist Queue leer ?
}
    temp:=q.FirstElPtr;                                   { NEIN: Größe des
}
    dec(Q.Size,temp^.Element.FS);                         { Frames wird von
}
                                                          { QSize abgezogen
}
    E:=temp^.Element;                                     { Inhalt d.
Elementes}
                                                          { wird übergeben.
}
    Q.FirstElPtr:=Q.FirstElPtr^.n;                        { Ptr wird umgesetzt
}
    dispose(temp);                                        { ,der Speicher
}
    if (Q.FirstElPtr=NIL) then Q.LastElPtr:=NIL;          { wird freigegeben
}
    RmEl:=true;                                           { Löschen
erfolgreich}
    dec(Q.Idx);                                           { Rahmenanz. dekrem.
}
  end
  else begin                                              { JA: Nichts tun
}
    RmEl:=false;                                          { Löschen nicht
}
  end;                                                    { erfolgreich!!
}
end;

(*-------------------------------------------------------*)
(* Bildung von Systemen aus mehreren Queues zur Dar-    *)
(* stellung von bestimmten Netzwerkkomponenten          *)
(*-------------------------------------------------------*)
procedure CutThroughMove(RandVal:Real;var FromQ,ToQ:Qtype);
var E:elmnt;
begin
  RandSeed:=FromQ.RSeed;                                  { Zufallsgenerator
}
```

```
                                                        { initialisieren,
}
  if (random < RandVal) then                            { ...und abfragen.
}
  if (FromQ.FirstElPtr<>NIL) then
  if ((ToQ.Size + FromQ.FirstElPtr^.Element.FS) < ToQ.MaxQSize) then begin
                                                        { Paßt Rahmen noch
in}
                                                        { den Empf.-
Speicher?}
    if (not ToQ.CalcFrame) then begin                   { Wird akt. Rahmen
}
                                                        { noch nicht bearb.?
}
      ToQ.CalcFrame:=true;                                    { JA: bearbeite
jetzt}
      E:=FromQ.FirstElPtr^.Element;                     { Erzeuge einen
neuen}
      dec(FromQ.Size,E.FS);                             { Eintrag in ZielQ.
}
      FromQ.FirstElPtr^.Element.FS:=0;                  { und ziehe Größe
von}
      InsEl(E,ToQ);                                     { Quellspeicher ab.
}
    end
    else begin                                          { Wird der Rahmen
}
                                                        { schon bearbeitet ?
}
      inc(ToQ.LastElPtr^.Element.FS,FromQ.FirstElPtr^.Element.FS);
                                                        { ,dann mach weiter
}
                                                        { (zähle
Zielspeicher}
                                                        { hoch, und Quellsp.
}
      dec(FromQ.Size,FromQ.FirstElPtr^.Element.FS);     { herunter)
}
      inc(ToQ.Size,FromQ.FirstElPtr^.Element.FS);
      FromQ.FirstElPtr^.Element.FS:=0;
    end;
    if FrameIsComplete(ToQ.LastElPtr^.Element) then begin
                                                        { Ist Frame komplett
                                                    }
                                                        { übertragen?
                                                    }
      RmEl(E,FromQ);                                    { JA: lösche Eintrag
}
                                                        { in QuellQueue.
}
      TokenIsFree[FromQ.Ring]:=true;                    { Gib Ring frei.
}
      ToQ.CalcFrame:=false;
    end;
  end;
  FromQ.RSeed:=RandSeed;                                { Zufallsgenerator  }
end;                                                    { zurücksetzen.     }


(*-------------------------------------------------------*)
(* Procedure zur Simulation des Store&Forward-Mechanis- *)
(* mus in Geräten (Switch/Bridge).                       *)
(*-------------------------------------------------------*)

procedure MoveFrameTo(RandVal:Real;var FromQ,ToQ:Qtype);
var E:elmnt;
begin
  RandSeed:=FromQ.RSeed;                                { Zufallsgenerator  }
                                                        { initialisieren    }
  if (random < RandVal) then                            { ...und abfragen   }
    if FrameIsComplete(FromQ.FirstElPtr^.Element) then begin
                                                        { Ist Frame komplett}
```

```
                                                    { angekommen?       }
      if ((ToQ.Size + FromQ.FirstElPtr^.Element.FS) < ToQ.MaxQSize) then
                                                    { und paßt es noch  }
                                                    { noch in den Puffer}
        if RmEl(E,FromQ) then begin                 { ,dann übertrage F.}

          if (not InsEl(E,ToQ)) then begin          { Nur interessant   }
            InsEl(E,FromQ);                         { für Multitasking- }
            inc(ToQ.Lost);                          { Systeme, falls ein}
          end                                       { Fehler auftritt.  }

          else if LOGGING then WriteLog(E);         { Schreibe Logdaten }
        end;                                        { wenn alles i.O ist}
end;
  FromQ.RSeed:=RandSeed;                            { Zufallsgenerator  }
end;                                                { zurücksetzen      }




(*-------------------------------------------------------*)
(* Procedure die einen Ring (ohne Prioritätsregelung)   *)
(* simuliert. Die Daten werden Byteweise (je nach Max-  *)
(* ThroughPut) von einer Queue in eine andere kopiert,  *)
(* falls der Ring/das Token (TokenIsFree) frei ist.     *)
(* Nach der Datenübertragung wird das Token wieder frei-*)
(* gegeben.                                             *)
(*-------------------------------------------------------*)
procedure MoveFrameThroughWire(RandVal:Real;var FromQ,ToQ:Qtype);
var E:elmnt;

begin
if (FromQ.FirstElPtr<>NIL) then                     { Ist Quelle leer?}
  if FrameIsComplete(FromQ.FirstElPtr^.Element) then begin      { Ist Rahmen
schon}
                                                    { komplett da ?   }
    if TokenIsFree[FromQ.Ring] then begin           { Ist das Token   }
                                                    { frei?           }
      RandSeed:=FromQ.RSeed;
      if (random < RandVal) then begin
        if ((ToQ.Size + FromQ.FirstElPtr^.Element.FS) < ToQ.MaxQSize) then
begin
                                                    { Paßt der Rahmen }
                                                    { noch in ziel-   }
                                                    { speicher?       }
          if (FromQ.FirstElPtr^.Element.FS > MAX_THROUGH_PUT ) then begin
            dec(FromQ.FirstElPtr^.Element.FS, MAX_THROUGH_PUT);
                                                    { JA: übertrage so}
            dec(FromQ.Size, MAX_THROUGH_PUT);               { viele Byte
wie  }
            E:= FromQ.FirstElPtr^.Element;          { erlaubt zu Ziel }
            E.FS:=MAX_THROUGH_PUT;
          end
          else begin                                { wurde Rahmenende}
            RmEl(E,FromQ);                                  { erreicht?
lösche}
          end;                                      { Eintrag aus     }
                                                    { Quellspeicher.  }
          if (InsEl(E,ToQ) = FALSE) then inc(ToQ.Lost);   { Nur interessant }
                                                    { für Multitasking}
                                                    { -Systeme        }
        end; {endif platz?}
        if FrameIsComplete(ToQ.LastElPtr^.Element) then   { wurde Rahmen    }
                                                    { komplett übertr.}
          TokenIsFree[FromQ.Ring]:=true                 { dann gib
Token  }
                                                    { wieder frei.    }
```

```
        else TokenIsFree[FromQ.Ring]:=false;
      end;     {if random?}
      FromQ.RSeed:=RandSeed;
    end;
  end
  else
  if (not FrameIsComplete(ToQ.LastElPtr^.Element)
  and (ToQ.LastElPtr<>NIL)) then begin
    if (FromQ.FirstElPtr^.Element.FS > MAX_THROUGH_PUT ) then begin
      if (( ToQ.Size + MAX_THROUGH_PUT ) < ToQ.MaxQSize) then begin
        dec(FromQ.FirstElPtr^.Element.FS,MAX_THROUGH_PUT);
        dec(FromQ.Size, MAX_THROUGH_PUT);
        E:= FromQ.FirstElPtr^.Element;
        E.FS:=MAX_THROUGH_PUT;
        inc(ToQ.Size, E.FS);
        inc(ToQ.LastElPtr^.Element.FS, E.FS);
      end      {if platz?}
    end
    else begin
      if (( ToQ.Size + FromQ.FirstElPtr^.Element.FS ) < ToQ.MaxQSize) then
begin
        E:=FromQ.FirstElPtr^.Element;
        RmEl(E,FromQ);
        inc(ToQ.Size, E.FS);
        inc(ToQ.LastElPtr^.Element.FS,E.FS);
      end;     {if platz?}
    end;
    if FrameIsComplete(ToQ.LastElPtr^.Element) then
      TokenIsFree[FromQ.Ring]:=true
    else
      TokenIsFree[FromQ.Ring]:=false;
  end;
end;

(*-------------------------------------------------------*)
(* Procedure die ein die Daten sendendes Gerät simuliert*)
(* Es erzeugt und sendet Datenrahmen an das als Empfäng-*)
(* er spezifizierte Gerät. Dabei wird darauf geachtet,  *)
(* daß ein LLC/ACK für die vorangegangenen Rahmen einge-*)
(* gangen ist bzw. der Wert für das WINDOW nicht über-  *)
(* schritten wird. Ist ein ACK-Rahmen empfangen worden, *)
(* so wird dieses bestätigt mittels eines LLC-Sequenz-  *)
(* Rahmens. Erst wenn dieses ACK-Frame empfangen wurde, *)
(* darf ein neues PAKET übertragen werden.              *)
(*-------------------------------------------------------*)

procedure Sender(var U:Netunit;var Q:QType);
var E:elmnt;
FrameSend:boolean;

begin
  FrameSend:=false;
  PAKET_SEND:=false;
  RandSeed:=U.GenRSeed;
  if ((random < U.GenPos) and (U.XmitedBytes < DataSize)) then begin
    case Q.LastFType of
      None        ,                                    { War vorher ge- }
      LLC         ,                                    { sendetes Frame }
      DataFirstMid:if (U.WantSeqLLC < U.WINDOW) then begin      { ein LLC-
Seq.od.}
                     if (U.LeftBytes > MFSize) then begin { DFM-Frame ??   }
                       E.FT:=DataFirstMid;               { Dann sende DFM-}
                       E.SA:=U.ADDR;                     { oder DOL-Frame!}
                       E.DA:=U.Dest;
                       E.FS:=MFSize + FHeaderSize        { Füge Header und}
                       •  FTrailerSize;                  { Trailer an.    }
                       E.CntrlBlck:=E.FS;                { FrameControll- }
                                                         { Feld ausfüllen.}
                       dec(U.LeftBytes,MFSize);          { Gesendete Bytes}
                                                         { von RestPaket  }
                                                         { abziehen.      }
                       inc(U.XmitedBytes,MFSize);
```

```
                    end
                    else begin                          { Ist RestPaket <}
                      E.FT:=DataOnlyLast;                      {
MaxFrameSize,so}
                      E.SA:=U.ADDR;                     { sende Rest als }
                      E.DA:=U.Dest;                     { DOL-Frame.     }
                      E.FS:=U.LeftBytes + FHeaderSize
                      •  FTrailerSize;
                      E.CntrlBlck:=E.FS;
                      inc(U.XmitedBytes,U.LeftBytes);
                      if (U.LeftData > PaketSize) then begin
                      U.LeftBytes:=PaketSize;
                      dec(U.LeftData,Paketsize);
                    end
                    else U.LeftBytes:=U.LeftData;
                    PAKET_SEND:=true;
                  end;
                  FrameSend:=true;
                  inc(U.WantSeqLLC)
                end;
    DataOnlyLast:if U.GotACK then begin                 { Wurde ein ACK  }
                  E.FT:=LLC;                            { empfangen, so  }
                  E.SA:=U.ADDR;                         { sende LLC-Seq- }
                  E.DA:=U.Dest;                         { Frame zurück.  }
                  E.FS:=LLC;
                  E.CntrlBlck:=LLC;
                  FrameSend:=true;
                end;
  end;
  if FrameSend then                                     { Frame wird in  }
    if (InsEl(E,Q)) then begin                          { Warteschlange  }
      Q.LastFType := Q.LastElPtr^.Element.FT;           { eingefügt.     }
      if ((Q.LastFType=DataFirstMid)
      or (Q.LastFType=DataOnlyLast)) then begin
        inc(U.first);
        if PAKET_SEND then begin
          U.GOTACK:=false;
          PAKET_SEND:=false;
        end;
      end;
    end;
  end;
  U.GenRSeed:=RandSeed;
end;

(*-------------------------------------------------------*)
(* Procedure die ein die Daten empfangendes Gerät simu- *)
(* liert. Es zählt die ankommenden Bytes und Rahmen.    *)
(* Ist ein Rahmen vollständig empfangen worden, wird je *)
(* nach Rahmentyp ein dementsprechendes Acknowledgement *)
(* zurück gesandt.                                      *)
(*-------------------------------------------------------*)

Procedure Empfaenger(var U:NetUnit;var InQ,OutQ:QType);
begin
  RandSeed:=U.ACKRSeed;                             { Vector für Zufallszahl}
                                                    { setzen (um Unabh. zu  }
                                                    { gewährleisten).       }
  if ((random < 0.6)
  and FrameIsComplete(InQ.FirstElPtr^.Element)) then begin
    if RmEl(El,InQ) then begin
      if ((El.FT = DataFirstMid)
      or (El.FT = DataOnlyLast)) then begin
        inc(U.XBs,(El.FS - (FHeaderSize + FTrailerSize)));
        inc(U.RecvdFrms);
        if (El.FT = DataOnlyLast) then begin        { Wurde ein Datenrahmen }
          ACKFrame.DA:=U.ADDR;                      { vom Typ DOL empfangen?}
          ACKFrame.SA:=U.Dest;
          InsEl(ACKFrame,OutQ);                     { Dann sende ACK zurück.}
        end;
        if (El.FT = DataFirstMid) then begin              { Wurde ein
Datenrahmen }
```

```
          LLCFrame.DA:=U.ADDR;                              { vom Typ DFM
empfangen?}
          LLCFrame.SA:=U.Dest;
          InsEl(LLCFrame,OutQ);                      { Dann sende LLC zurück.}
        end;
      end;
    end;
  end;
  U.ACKRSeed:=RandSeed;                                     { Vector speichern
}
end;

(*-------------------------------------------------------*)
(* Anzeigen des Momentanen Systemstatus‘ während des    *)
(* Durchlaufs (QLength)                                  *)
(*-------------------------------------------------------*)

procedure ShowQLength(Q:QType;color:byte);
var length:word;
begin
  length:=trunc((Q.Size * 100) / Q.MaxQSize);               { Prozentuale
Darstellg.}
  if (length > 0) then begin                         { der Speicherauslastg. }
    textcolor(7);
    textbackground(color);
    write(length:3,‘ % ,);
    textcolor(14);
    length:=trunc(length * (MaxLength / 100));       { Skalierung vornehmen. }
    for i:=1 to length do write(,¦‘);                { Ausgabe in Balkenform }
    for i:=length to MaxLength do write(, ,);              { Rest mit
Leerzeichen  }
    textbackground(bkcolor);                         { füllen.               }
    write(, Bytes:‘,Q.Size:6);                       { Speicherauslastung in }
    writeln;                                         { Byte.                 }
  end
  else begin                                         { Wenn proz.Auslastg.   }
    textcolor(7);                                    { zu gering für Darst.  }
    textbackground(color);                                 { verkürze
Ausgabealgo- }
    write(length:3,‘ % ,);                                 { rithmus.
}
    for i:=0 to MaxLength do write(, ,);
    textbackground(bkcolor);
    write(, Bytes:‘,Q.Size:6);
    writeln;
  end;
  textcolor(4);
end;

(*-------------------------------------------------------*)
(* Auswerten der an das Program übergebenen Parameter   *)
(*-------------------------------------------------------*)

procedure ParsParam(nr:byte);
var pp:string;
begin
  err:=0;                                                  { err: Hilfsvariable}
  pp:=paramstr(nr);                                        { zur Fehlererken-  }
  case pp[1] of                                            { nung bei Datenkon-}
                                                           { vertierung        }
    ,m‘,‘M‘:begin
              delete(pp,1,1);                              { Die an das Pro-   }
              val(pp,MaxFrameSize,err);                          { gramm
übergebenen }
            end;                                           { Parameter auslesen}
    ,f‘,‘F‘:begin                                          { und in den ent-   }
              delete(pp,1,1);                              { sprechenden Vari- }
              val(pp,PaketSize,err);                       { ablen speichern.  }
            end;                                           { m - MaxFrameSize  }
                                                           { f - PaketSize     }
    ,s‘,‘S‘:SHOWLENGTH:=true;                              { s - SHOWLENGTH    }
                                                           {     (Zeigen der   }
```

```
                                                          {     animierten    }
                                                          {     Simulation)   }
    ,t‘,‘T‘:begin                                         { t - TraceStep     }
              TRACE:=true;
              delete(pp,1,1);
              val(pp,TraceStep,err);
            end;
    ,d‘,‘D‘:begin                                         { d - DataSize      }
              delete(pp,1,1);
              val(pp,DataSize,err);
            end;
    ,i‘,‘I‘:begin                                         { i - Pfad/Name für }
              delete(pp,1,1);                             {     INI-Datei     }
              INIFile:=pp;
              UseINI:=true;
            end;
    ,o‘,‘O‘:begin                                         { o – Pfad/Name für }
              delete(pp,1,1);                             {     Ausgabedatei  }
              OutFile:=pp;
              PRNSTAT:=true;
            end;
    ,h‘,‘H‘:begin
              writeln(,MadQ Version 0.42 ©1996‘);
              writeln(,Usage:‘);
              writeln(,       MADQ.EXE [Options1] [Option2] ...‘);
              writeln(,Possible options are the following.# stands for a
                       value.‘);
              writeln(,   M#         - Maximum Frame Size,that could be
                       transmited‘);
              writeln(,   F#         - Frame Size used by higher layer‘);
              writeln(,   S          - Show animated queue status‘);
              writeln(,   T#         - Trace mode (Step Width)‘);
              writeln(,   D#         - Data (byte) to be transfered‘);
              writeln(,   H          - Help informations‘);
              writeln(,   I<name>    - use INI-file <name>‘);
              writeln(,   O<name>    - write output to file <name>‘);
              writeln;
              writeln(,Example: MadQ m4096 f1024 d6553500 s iMADQ.INI‘);
              writeln;
              halt;
            end;
  end;
  if err<>0 then begin                                                {
Fehlerbehandlung }
    writeln(,Error:Invalid Parameter #‘,nr,‘ given...‘);  { Ausgabe der Feh- }
    halt;                                                       { lermeldung;
Pro- }
  end;                                                          { gramm
beenden!   }
end;

(*-------------------------------------------------------*)
(* Ausgabe der Messwerte in ASCII-Datei (in Tabellen-   *)
(* form                                                  *)
(*-------------------------------------------------------*)
procedure PrintStatus(U:NetUnit;datei:string);
var f:text;
begin
  if Exists(datei) then begin                             { Existiert File?  }
    assign(f,datei);                                      { JA:Daten anhängen}
    append(f);                                            { Messwerte ausgebn}
    writeln(f,PaketSize:5,‘:  ,,zeittakt:10,‘ ,,U.rate,‘
              ,,‘XmitedBytes:‘,Unit1.XBs);
    close(f);
  end
  else begin                                              { NEIN:File er-    }
    assign(f,datei);                                      { stellen.         }
    rewrite(f);                                           { Messwerte ausgebn}
    writeln(f,PaketSize:5,‘:  ,,zeittakt:10,‘ ,,U.rate,‘
            ,,‘XmitedBytes:‘,Unit1.XBs);
    close(f);
  end;
```

```
end;

(*-----------------------------------------------------------------------*)

procedure Ausgabe(var U:NetUnit);
begin
  U.Rate:=((RealVal(U.XBs) * RealVal(1024)) / (ZeitTakt * TimeFactor))*14-
190000;
  If (U.Rate<0) then U.Rate:=0;                              { Umrechnung der
}
  GotoXY(1,1);                                               { werte in reale
}
  if SHOWLENGTH then begin                                         {
Übertragungsrate   }
    ShowQLength(Q0,1);                                       { und Ausgabe.
}
    ShowQLength(Q1,1);                                       { zusätzlich Ausgabe
}
    ShowQLength(Q2,1);                                       { der Pufferfüllung
}
    ShowQLength(Q3,1);                                       { der einzelnen War-
}
    ShowQLength(Qb0,4);                                      { teschlangen für d.
}
    ShowQLength(Qb1,4);                                      { verschiedenen
Ports}
    ShowQLength(Qb2,4);
    ShowQLength(Qb3,4);
    Textcolor(7);
    writeln(, Rate:   ,,trunc(U.Rate):8,'.',trunc(frac(U.rate)*10):1,
      , Byte/s  Timer: ,,ZeitTakt:14,'   Xmitted Frames: ,,U.RecvdFrms:10);
    writeln(, PaketSize: ,,PaketSize:7,' Byte    MaxFrameSize:
,,MaxFrameSize:7,
      ,   Left Frames: ,,(XmitFrameCount-U.first):13);
    writeln(, Xmited Bytes:',U.XBs:10);
  end
  else
    writeln((U.XBs*100) div DataSize,'% of ,,DataSize,' Byte übertragen.');
end;

(*-------------------------------------------------------*)
(* Initialisierung des Systems                           *)
(*-------------------------------------------------------*)
procedure InitQstatus;
procedure InitQ(var Qx:Qtype);                               { Initialisierung
}
begin                                                        { der einzelnen War-
}
  with Qx do begin                                           { teschlangen.
}
    Lost:=0;                                                 { Verlorene
Rahmen=0.}
    Idx:=0;                                                  { Rahmen in Queue =0
}
    Size:=0;                                                 { Byte in Queue = 0
}
    FirstElPtr:=NIL;                                         { Pointer Initialis.
}
    LastElPtr:=NIL;
    LastFType:=none;                                         { Letzter
behandelter}
                                                             { Rahmentyp = KEINER
}
    CalcFrame:=false;                                        { Wird Rahmen bearb-
}
                                                             { eitet ?: NEIN
}
  end;
end;

begin                                                        { Anfang Initiali-
}
```

```
                                                        { sierung des
Systems}
  Mark(QTop);                                                 { Stack
initialis.   }
  TraceStep:=1;                                         { Defaultwert für
}
                                                        { Tracing=1
(ständig)}
  Unit1.GOTACK:=true;                                   { System freimachen
}
  for i:=1 to 255 do TokenIsFree[i]:=true;              { für erstes Senden.
}
  with LLCFrame do begin                                { LLC-Rahmentyp
}
    FT:=LLC;                                            { initialisieren
}
    FS:=LLC;                                            { (Größe,Type,Con-
}
    CntrlBlck:=LLC;                                     { trollwert)
}
  end;
  with ACKFrame do begin                                { ACK-Rahmentyp
}
    FT:=ACK;                                            { initialisieren
}
    FS:=ACK;                                            { {wie LLC-Rahmen)
}
    CntrlBlck:=ACK;
  end;
  InitQ(Q0);                                            { Einzelne
Warteschl.}
  InitQ(Q1);                                            { initialisieren
}
  InitQ(Q2);                                            { (Sender,
Empfänger)}
  InitQ(Q3);
  InitQ(Qb3);
  InitQ(Qb2);
  InitQ(Qb1);
  InitQ(Qb0);
  ZeitTakt :=0;                                         { Timer-Tic auf 0
}
  Unit1.RecvdFrms:=0;                                   { Vom Empfänger
empf.}
                                                        { DatenRahmen auf 0
}
  Unit1.WantSeqLLC:=0;                                  { Ausstehende Con-
}
                                                        { troll-Rahmen auf 0
}
  Unit1.First:=0;                                       { Standardwerte neh-
}
  SHOWLENGTH:=false;                                    { men,falls keine
}
  PRNSTAT   :=false;                                    { Parameter angege-
}
  UseINI    :=false;                                    { ben wurden.
}
  TRACE     :=false;
  XmitFrameCount:=0;
  DataSize      :=2147480000;
  PaketSize     :=4096;
  MaxFrameSize  :=4096;
  if (paramcount > 0) then                                    { Werden
Parameter   }
    for i:=1 to Paramcount do ParsParam(i);             { übergeben, dann
}
                                                        { einlesen...
}

{ Berechnung der Größe und Anzahl der zu übertragenden Datenrahmen       }
```

```
  if ((MaxFrameSize - (FHeaderSize + FTrailerSize)) < PaketSize) then
    MFSize:=(MaxFrameSize - (FHeaderSize + FTrailerSize))
                                                        { Ist die Paketgröße}
                                                        { größer als die
                                                  max.}
                                                        { Rahmengröße, so
}
                                                        { unterteile Pakete
}
                                                        { in Rahmen
max.Größe}
  else                                                  { SONST: Rahmengröße
}
    MFSize:=PaketSize;                                  { ist gleich der
}
                                                        { Paketgröße
}
  FrameNumb:=(PaketSize div MFSize);
  if ((PaketSize mod MFSize)<>0)then                    { Bleibt ein Rest-
}
    inc(FrameNumb);                                     { Rahmen so erhöhe
d.}
  XmitFrameCount:=(DataSize div PaketSize);             { Anz. Der zu über-
}
  XmitFrameCount:=XmitFrameCount * FrameNumb;                 { tragenden
Daten-   }
  RestPaketSize:=DataSize mod PaketSize;                { rahmen.
}
  if (RestPaketSize<>0) then begin
  RestFrames:=RestPaketSize div MFSize;
  if ((RestPaketSize mod MFSize)<>0) then inc(RestFrames);
  inc(XmitFrameCount,RestFrames);
end;
XmitMode:=1;                                            { Store&Forward-Mode
}
Unit1.LeftBytes:=PaketSize;
Unit1.Leftdata:=DataSize;
Unit1.WINDOW:=3;                                        { Rahmenfenster = 3
}
val(GetIniData(IniFile,'Station1.Buffer1Size'),         { Speichergrößen der
}
    Q0.MaxQSize,err);                                   { einzelnen Ports
der}
val(GetIniData(IniFile,'Station1.Buffer2Size'),         { der
Netzwerkknoten.}
    Q1.MaxQSize,err);
val(GetIniData(IniFile,'Station1.Buffer3Size'),
    Q2.MaxQSize,err);
val(GetIniData(IniFile,'Station1.Buffer4Size'),
    Q3.MaxQSize,err);
val(GetIniData(IniFile,'Station1.Buffer1Size'),
    Qb0.MaxQSize,err);
val(GetIniData(IniFile,'Station1.Buffer2Size'),
    Qb1.MaxQSize,err);
val(GetIniData(IniFile,'Station1.Buffer3Size'),
    Qb2.MaxQSize,err);
val(GetIniData(IniFile,'Station1.Buffer4Size'),Qb3.MaxQSize,err);
val(GetIniData(IniFile,'Station1.Possibility1'),        { Initialisierung
der}
    Q0.Possblty,err);                                   { Paketgen.- und Ver-
}
val(GetIniData(IniFile,'Station1.Possibility2'),        { arbeitungswahrsch.
}
    Q1.Possblty,err);                                   { für die einzelnen
}
val(GetIniData(IniFile,'Station1.Possibility3'),        { Geräte.
}
    Q2.Possblty,err);
val(GetIniData(IniFile,'Station1.Possibility4'),
    Q3.Possblty,err);
val(GetIniData(IniFile,'Station1.Possibility1b'),Qb0.Possblty,err);
val(GetIniData(IniFile,'Station1.Possibility2b'),Qb1.Possblty,err);
```

```
val(GetIniData(IniFile,‘Station1.Possibility3b‘),Qb2.Possblty,err);
val(GetIniData(IniFile,‘Station1.Possibility4b‘),Qb3.Possblty,err);
val(GetIniData(IniFile,‘Station1.Window‘),                { Fenstergröße = 2  }
    Unit1.WINDOW,err);
val(GetIniData(IniFile,‘Station1.Ring1‘),Q0.Ring,err);    { Ringzugehörigkeit }
val(GetIniData(IniFile,‘Station1.Ring2‘),Q1.Ring,err);    { und Topologie     }
val(GetIniData(IniFile,‘Station1.Ring3‘),Q2.Ring,err);
val(GetIniData(IniFile,‘Station1.Ring4‘),Q3.Ring,err);
val(GetIniData(IniFile,‘Station1.Ring1b‘),Qb0.Ring,err);
val(GetIniData(IniFile,‘Station1.Ring2b‘),Qb1.Ring,err);
val(GetIniData(IniFile,‘Station1.Ring3b‘),Qb2.Ring,err);
val(GetIniData(IniFile,‘Station1.Ring4b‘),Qb3.Ring,err);
val(GetIniData(IniFile,‘System.TimeFactor‘),              { Umrechnungsfaktor }
    TimeFactor,err);                                      { Sim. zu Realsystem}
val(GetIniData(IniFile,‘System.GenFramePoss‘),Unit1.GenPos,err);
val(GetIniData(IniFile,‘System.MaxThroughPut‘),           { Maximale Daten-   }
    MAX_THROUGH_PUT,err);                                 { durchsatz des Ring}
val(GetIniData(IniFile,‘System.FixedRandSeed‘),           { Sollen Standard-  }
    FixedSeed,err);                                       { werte f. d. Simu- }
                                                          { lation genutzt    }
                                                          { werden?1=Ja/0=Nein}
val(GetIniData(IniFile,‘System.RingAnzahl‘),RingAnzahl,err);
val(GetIniData(IniFile,‘System.LastRechner‘),Lastrechner,err);
val(GetIniData(IniFile,‘System.TransferMode‘),            { legt Übertragungs-}
    XmitMode,err);                                        { modus fest.       }

if (FixedSeed=1) then begin                               { Falls Standard-   }
  val(GetIniData(IniFile,‘System.GenRandSeed‘),           { werte benutzt wer-}
      Unit1.GenRSeed,err);                                      { sollen, hole
die  }
  val(GetIniData(IniFile,‘System.ACKRandSeed‘),           { entsprechenden    }
      Unit1.ACKRSeed,err);                                      { Werte aus der
INI-}
  val(GetIniData(IniFile,‘System.Q1RandSeed‘),            { Datei.            }
      Q0.RSeed,err);
  val(GetIniData(IniFile,‘System.Q2RandSeed‘),
      Q1.RSeed,err);
  val(GetIniData(IniFile,‘System.Q3RandSeed‘),
      Q2.RSeed,err);
  val(GetIniData(IniFile,‘System.Qb1RandSeed‘),
      Qb0.RSeed,err);
  val(GetIniData(IniFile,‘System.Qb2RandSeed‘),
      Qb1.RSeed,err);
  val(GetIniData(IniFile,‘System.Qb3RandSeed‘),
      Qb2.RSeed,err);
end
else begin                                                { Sonst initiali-  }
  Randomize;                                              { siere mittels    }
  Unit1.GenRSeed:=RandSeed;                               { Zufallsgenerator }
  Randomize;
  Unit1.ACKRSeed:=RandSeed;
  Randomize;
  Q0.RSeed:=RandSeed;
  Randomize;
  Q1.RSeed:=RandSeed;
  Randomize;
  Q2.RSeed:=RandSeed;
  Randomize;
  Qb0.RSeed:=RandSeed;
  Randomize;
  Qb1.RSeed:=RandSeed;
  Randomize;
  Qb2.RSeed:=RandSeed;
end;
Unit1.XBs:=0;                                                   { Anzahl
übertra-  }
Unit1.XmitedBytes:=0;                                     { gener Bytes=0.    }
Unit1.ADDR:=1;
Unit1.Dest:=2;
if Lastrechner>0 then for i:=1 to Lastrechner do begin
  InitQ(Ql0[i]);                                          { Initialisierung   }
  InitQ(Ql1[i]);                                          { der Warteschlangen}
```

```
  InitQ(Ql2[i]);                                          { der einzelnen     }
  InitQ(Ql3[i]);                                          { Lastrechner.      }
  InitQ(Qlb3[i]);
  InitQ(Qlb2[i]);
  InitQ(Qlb1[i]);
  InitQ(Qlb0[i]);
  Last[i].GOTACK:=true;                                   { Setzen der anderen}
  Last[i].XmitedBytes:=0;                                 { Parameter d. Ports}
  Last[i].XBs:=0;                                         { wie Adresse usw.  }
  Last[i].ADDR:=i*2+1;
  Last[i].Dest:=i*2+2;
  Last[i].First:=0;
  Last[i].WantSeqLLC:=0;
  Last[i].RecvdFrms:=0;
  Randomize;                                              { Festlegen der Rah-}
  Last[i].GenRSeed:=RandSeed;                             { mengenerierungs-  }
  Randomize;                                              { und Verarbeitungs-}
  Last[i].ACKRSeed:=RandSeed;                             { wahrscheinlichkei-}
  Randomize;                                              { ten.
  Ql0[i].RSeed:=RandSeed;
  Randomize;
  Ql1[i].RSeed:=RandSeed;
  Randomize;
  Ql2[i].RSeed:=RandSeed;
  Randomize;
  Qlb0[i].RSeed:=RandSeed;
  Randomize;
  Qlb1[i].RSeed:=RandSeed;
  Randomize;
  Qlb2[i].RSeed:=RandSeed;
  val(GetIniData(IniFile,‘Station1.Window‘),              { Setzen der Rahmen-}
      Last[i].WINDOW,err);                                      { fenstergröße
und  }
  val(GetIniData(IniFile,‘Station1.Buffer1Size‘),         { der Speichergrößen}
      Ql0[i].MaxQSize,err);
  val(GetIniData(IniFile,‘Station1.Buffer2Size‘),
      Ql1[i].MaxQSize,err);
  val(GetIniData(IniFile,‘Station1.Buffer3Size‘),
      Ql2[i].MaxQSize,err);
  val(GetIniData(IniFile,‘Station1.Buffer4Size‘),
      Ql3[i].MaxQSize,err);
  val(GetIniData(IniFile,‘Station1.Buffer1Size‘),Qlb0[i].MaxQSize,err);
  val(GetIniData(IniFile,‘Station1.Buffer2Size‘),Qlb1[i].MaxQSize,err);
  val(GetIniData(IniFile,‘Station1.Buffer3Size‘),Qlb2[i].MaxQSize,err);
  val(GetIniData(IniFile,‘Station1.Buffer4Size‘),Qlb3[i].MaxQSize,err);
  val(GetIniData(IniFile,‘Station1.Possibility1‘),Ql0[i].Possblty,err);
  val(GetIniData(IniFile,‘Station1.Possibility2‘),Ql1[i].Possblty,err);
  val(GetIniData(IniFile,‘Station1.Possibility3‘),Ql2[i].Possblty,err);
  val(GetIniData(IniFile,‘Station1.Possibility4‘),Ql3[i].Possblty,err);
  val(GetIniData(IniFile,‘Station1.Possibility1b‘),Qlb0[i].Possblty,err);
  val(GetIniData(IniFile,‘Station1.Possibility2b‘),Qlb1[i].Possblty,err);
  val(GetIniData(IniFile,‘Station1.Possibility3b‘),Qlb2[i].Possblty,err);
  val(GetIniData(IniFile,‘Station1.Possibility4b‘),Qlb3[i].Possblty,err);
  val(GetIniData(IniFile,‘Station1.Ring1‘),Ql0[i].Ring,err);
  val(GetIniData(IniFile,‘Station1.Ring2‘),Ql1[i].Ring,err);
  val(GetIniData(IniFile,‘Station1.Ring3‘),Ql2[i].Ring,err);
  val(GetIniData(IniFile,‘Station1.Ring4‘),Ql3[i].Ring,err);
  val(GetIniData(IniFile,‘Station1.Ring1b‘),Qlb0[i].Ring,err);
  val(GetIniData(IniFile,‘Station1.Ring2b‘),Qlb1[i].Ring,err);
  val(GetIniData(IniFile,‘Station1.Ring3b‘),Qlb2[i].Ring,err);
  val(GetIniData(IniFile,‘Station1.Ring4b‘),Qlb3[i].Ring,err);
  val(GetIniData(IniFile,‘System.GenFramePoss‘),Last[i].GenPos,err);
  Last[i].LeftBytes:=PaketSize;
  Last[i].LeftData:=DataSize;
end;
LogFile:=GetIniData(IniFile,‘System.LogFile‘);
LOGGING:=(GetIniData(IniFile,‘System.Logging’)=‘1‘);
end;

(*****************************************************************)
(*                          HAUPTPROGRAMM                        *)
(*****************************************************************)
```

```
begin
  GetDefCursor;
  SetCursorType(8,0);
  textbackground(bkcolor);
  clrscr;
  gotoxy(1,23);
  writeln(,MadQ Version 0.50 ©1996‘);
  InitQStatus;                                          { Aufruf der System-
}
                                                        { initialisierungs-
}
                                                        { routinen.
}
  while (( not keypressed)                              { Solange keine Tas-
}
                                                        { te gedrückt und
}
  and (Unit1.RecvdFrms < XmitFrameCount)) do begin      { Sim. nicht
komplett}
                                                        { führe Sim. durch.
}
    Sender(Unit1,Q0);                                   { Generiere Rahmen
}
    if RingAnzahl=1 then                                { Falls Sender und
}
                                                        { Empfänger in einem
}
                                                        { Ring:
}
      MoveFrameThroughWire(Q0.Possblty,Q0,Q3)                 { übergebe
direkt an }
                                                        { Empfänger.
}
    else begin                                          { sonst übergib Rah-
}
      MoveFrameThroughWire(Q0.Possblty,Q0,Q1);          { men an Bridge.
}
      if XmitMode=1 then                                { Übertragung via
S&F}
        MoveFrameTo(Q1.Possblty,Q1,Q2)
      else                                              { bzw. CT
}
        CutThroughMove(Q1.Possblty,Q1,Q2);
      MoveFrameThroughWire(Q2.Possblty,Q2,Q3);
    end;
    Empfaenger(Unit1,Q3,Qb3);                           { Verarbeite Rahmen
}
                                                        { in Empfänger.
}
    if RingAnzahl=1 then
      MoveFrameThroughWire(Qb3.Possblty,Qb3,Qb0)
    else begin
      MoveFrameThroughWire(Qb3.Possblty,Qb3,Qb2);
      if XmitMode=1 then
        MoveFrameTo(Qb2.Possblty,Qb2,Qb1)
      else
        CutThroughMove(Qb2.Possblty,Qb2,Qb1);
      MoveFrameThroughWire(Qb1.Possblty,Qb1,Qb0);
    end;
    RandSeed:=Qb0.RSeed;
    if ((random < 0.5)
    and FrameIsComplete(Qb0.FirstElPtr^.Element)) then begin
      if RmEl(El,Qb0) then begin
        if (El.FT = ACK) then begin
          Unit1.GOTACK:=true;
          dec(Unit1.WantSeqLLC);
        end;
        if (El.FT = LLC) then dec(Unit1.WantSeqLLC);
      end;
    end;
    Qb0.RSeed:=RandSeed;
```

```
    if Lastrechner>0 then for i:=1 to Lastrechner do begin      { Selbiges wie
oben  }
      Sender(Last[i],Ql0[i]);                                   { für Lastrechner.
}
      if RingAnzahl=1 then
        MoveFrameThroughWire(Ql0[i].Possblty,Ql0[i],Ql3[i])
      else begin
        MoveFrameThroughWire(Ql0[i].Possblty,Ql0[i],Ql1[i]);
        if XmitMode=1 then
          MoveFrameTo(Ql1[i].Possblty,Ql1[i],Ql2[i])
        else
          CutThroughMove(Ql1[i].Possblty,Ql1[i],Ql2[i]);
        MoveFrameThroughWire(Ql2[i].Possblty,Ql2[i],Ql3[i]);
      end;
      Empfaenger(Last[i],Ql3[i],Qlb3[i]);
      if RingAnzahl=1 then
        MoveFrameThroughWire(Qlb3[i].Possblty,Qlb3[i],Qlb0[i])
      else begin
        MoveFrameThroughWire(Qlb3[i].Possblty,Qlb3[i],Qlb2[i]);
        if XmitMode=1 then MoveFrameTo(Qlb2[i].Possblty,Qlb2[i],Qlb1[i])
        else CutThroughMove(Qlb2[i].Possblty,Qlb2[i],Qlb1[i]);
        MoveFrameThroughWire(Qlb1[i].Possblty,Qlb1[i],Qlb0[i]);
      end;
      RandSeed:=Qlb0[i].RSeed;
      if ((random < 0.5)
      and FrameIsComplete(Qlb0[i].FirstElPtr^.Element)) then begin
        if RmEl(El,Qlb0[i]) then begin
          if (El.FT = ACK) then begin
            Last[i].GOTACK:=true;
            dec(Last[i].WantSeqLLC);
          end;
          if (El.FT = LLC) then dec(Last[i].WantSeqLLC);
        end;
      end;
      Qlb0[i].RSeed:=RandSeed;
    end;
    inc(ZeitTakt);                                              { Systemzeit aktuali-
}
                                                                { sieren.
}
    Ausgabe(Unit1);                                             { aktuelle Daten aus-
}
                                                                { geben.
}
    if (TRACE                                                   { Falls getracet wer-
}
    and (((ZeitTakt-1) mod TraceStep)=(TraceStep-1))) then      { den soll und
Marke }
      readln;                                                   { erreicht,
warte auf}
  end;                                                          { Tasteneingabe.
}
  if PRNSTAT then
    Printstatus(Unit1,OutFile);                                 { Ausgabe in Datei ,
}
                                                                { falls Logging ein-
}
                                                                { schaltet.
}
  if keypressed then                                            { Falls Tastendruck,
}
    readkey;                                                    { Taste lesen.
}
  Release(QTop);
  textbackground(0);
  textcolor(7);
  clrscr;
  SetDefCursor;
end.
(*************** ENDE ** MadQ.PAS ***********************************)
```

# A2. Initialisierungsdatei (Madq.ini)

Beispiel einer Initialisierungsdatei für den Netzwerksimulator MadQ, wie sie unter anderem auch in einzelnen Meßreihen benutzt wurde. Der Abschnitt `[System]` enthält die Daten für das Gesamtsystem, wie Übertragungsmodus, Anzahl der Lastrechner und Ringe, Umrechnungsfaktoren von Simulations- zu Realsystem, Standardwerte für Rahmengenerierungs- und Verarbeitungswahrscheinlichkeiten und Pfade bzw. Namen der LOG-Dateien.

```
[System]
TransferMode=1
Logging=0
LogFile=D:\MadQ\Test\nix1_2.lgn
LastRechner=3
RingAnzahl=2
MaxThroughPut=3
FixedRandSeed=0
TimeFactor=0.02
GenFramePoss=0.42
GenRandSeed=85485
ACKRandSeed=69451342
Q1RandSeed=4214760436
Q2RandSeed=72343
Q3RandSeed=9834844356
Qb1RandSeed=42159476
Qb2RandSeed=45
Qb3RandSeed=3654844
```

In `[Netzwerk]` können zusätzliche Angaben zu dem Netzwerk angegeben werden.

```
[Netzwerk]
Typ=TokenRing
Stationen=11
Ringe=3
Struktur=(1.3 - 2.5),(2.6 - 3.1)
Protokoll=Netbios
```

In `[Stationx]` werden die Angaben über die verschiedenen Netzwerkknoten eingetragen. Zu diesen Angaben zählen Name, Adresse, Speichergrößen, Anzahl der Ports, Verarbeitungsleistung und Einbindung der einzelnen Ports in die Netztopologie.

```
[Station1]
Name=Wilma
```

```
Typ=Sender
Queues=4
Window=3
Buffer1Size=128000
Buffer2Size=512000
Buffer3Size=512000
Buffer4Size=128000
Possibility1=0.5
Possibility2=1
Possibility3=1
Possibility4=0.5
Possibility1b=0.5
Possibility2b=1
Possibility3b=1
Possibility4b=0.5
Ring1=1
Ring2=1
Ring3=2
Ring4=2
Ring1b=2
Ring2b=2
Ring3b=1
Ring4b=1
```

# Erklärung

Ich versichere, daß ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Ort Datum Unterschrift