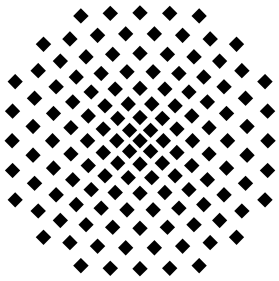


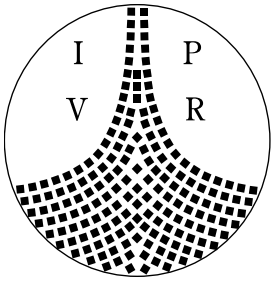
Universität Stuttgart
Fakultät Informatik

Prüfer:	Prof. Dr. K. Rothermel
Betreuer:	Dipl.-Inf. Daniel Herrscher
Beginn am:	15.04.2002
Beendet am:	11.10.2002
CR-Nummer:	C.2.5, D.4.4, I.6.3, I.6.8

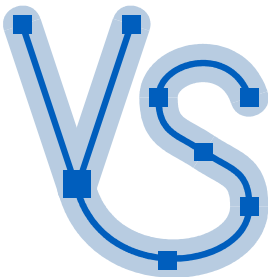
Diplomarbeit Nr. 2000

Emulationskonzepte für Netze mit gemeinsamem Medium

Steffen Maier



Institut für Parallele und
Verteilte Systeme (IPVS)
Abteilung Verteilte Systeme
Breitwiesenstr. 20–22
70565 Stuttgart



Zusammenfassung

Gegenstand derzeitiger und sicherlich auch zukünftiger Forschungsarbeiten ist die Entwicklung verteilter Anwendungen sowie die Entwicklung von Netzwerkprotokollen. Um die Auswirkungen verschiedener Netzumgebungen auf Anwendungen und Netzwerkprotokolle zu testen, ist neben mathematischer Analyse und Simulation auch die vergleichende Leistungsmessung der entwickelten Software unverzichtbar. Als Alternative zur Messung in realer Umgebung bietet sich die Emulation von Netzumgebungen an oder stellt unter Umständen sogar die einzige Testmöglichkeit dar.

Das NET-Projekt (Network Emulation Testbed) an der Abteilung Verteilte Systeme hat eine flexible, umfassende Netzemulationsumgebung für reproduzier- und vergleichbare Leistungsmessungen zum Ziel. Zur Emulation von Punkt-zu-Punkt Verbindungen existiert bereits das Werkzeug NETShaper. Zur Erfüllung des Aspekts der umfassenden Emulationsumgebung werden auch Emulationsverfahren für Netze mit gemeinsamem Medium benötigt. Es war nun Aufgabe dieser Diplomarbeit, ein Emulationskonzept für kabelgebundene Netze mit gemeinsamem Medium zu entwickeln und prototypisch umzusetzen.

Die vorliegende Ausarbeitung beginnt mit einer Darstellung der Motivation und Beschreibung der Aufgabenstellung. Anschließend folgt ein Überblick über die wichtigsten technischen Grundlagen, auf denen diese Arbeit basiert. Außerdem werden verwandte Arbeiten untersucht und gegen die vorliegende abgegrenzt.

Nach der Aufstellung von Evaluationskriterien für Emulationskonzepte werden erarbeitete Konzepte verschiedener Ausprägungen vorgestellt. Basierend auf der Evaluation der vorgestellten Konzepte wird ein Ethernet-Emulationskonzept als erfolgversprechendes ausgewählt. Für das ausgewählte Konzept wird die Spezifikation, der Entwurf und die Realisierung in Form eines Linux-Kernel-Moduls beschrieben. Durchgeführte Messungen zur Untersuchung der Realitätsnähe der Implementierung und deren Ergebnisse werden im Anschluß erläutert.

Im Ausblick folgt die Vorstellung von möglichen Erweiterungen und Verbesserungen der umgesetzten Ethernet-Emulation sowie die Identifizierung zukünftiger Arbeiten. Den Abschluß der Ausarbeitung bilden die aus der Arbeit gezogenen Schlußfolgerungen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einführung

1.1 Motivation

Gegenstand derzeitiger und sicherlich auch zukünftiger Forschungsarbeiten ist die Entwicklung verteilter Anwendungen sowie die Entwicklung von Netzwerkprotokollen. Verteilte Anwendungen werden beispielsweise mit Hilfe von Middleware in Form von Corba, Web Services oder der Java Enterprise Architektur realisiert. Zur Kommunikation benötigt die Middleware wiederum ein Netzwerk. Neben den seit langem eingesetzten privaten LANs (Local Area Networks) und den später sich schnell verbreitenden WAN-Verbindungen (Wide Area Network) wird zunehmend auch der Einsatz von Funknetzen, die nicht mehr kabelgebunden sind, interessant.

Um die Auswirkungen verschiedener Netzumgebungen auf verteilte Anwendungen und Netzwerkprotokolle zu analysieren, stehen zu Beginn einer Entwicklung mathematische Analysen und Simulationen im Vordergrund. Dazu werden anhand von vereinfachenden Modellen durch Auswahl verschiedener Modellparameter die Auswirkungen auf das Verhalten der zu prüfenden Software untersucht. Später werden im Laufe der Entwicklung reale Implementierungen benötigt. Unverzichtbar ist die vergleichende Leistungsmessung der entwickelten Software. Sie erlaubt gegenüber Analyse und Simulation detailliertere Aussagen über das tatsächliche Verhalten der Software. Die Messungen lassen sich beispielsweise innerhalb einer realen Netzumgebung durchführen. Allerdings ist diese Vorgehensweise eingeschränkt durch die Verfügbarkeit und Skalierbarkeit einer solchen Netzumgebung. Während beispielsweise eine Simulation durch Abstriche in ihrer Ablaufgeschwindigkeit größere Szenarien nachbilden kann, ist dies bei Messungen in einer Realumgebung nicht möglich. Bei letzteren läuft der Versuch zwangsläufig in Echtzeit ab und die Szenariogröße ist durch die verfügbare Hardware, z.B. die Anzahl der Knoten und Netzverbindungen, vorgegeben.

Eine flexible, umfassende Netzumgebung für reproduzier- und vergleichbare Leistungsmessungen wird also benötigt. Innerhalb des NET-Projekts (Network Emulation Testbed) an der Abteilung Verteilte Systeme wird eine solche Umgebung aufgebaut. Sie besteht aus einem Rechner-Cluster als Hardware-Plattform sowie Software-Werkzeugen zur Emulation von Netzen und Steuerung des Emulationsablaufes. Zur Emulation von Punkt-zu-Punkt-Netzverbindungen existiert im Rahmen des Projekts bereits das Emulationswerkzeug NETShaper. Es dient dazu, anhand der spezifizierten Sicherungsschichtparameter Übertragungsgeschwindigkeit, Ausbreitungsverzögerung und einer Rahmenfehlerrate, Verbindungen beispielsweise aus dem WAN-Bereich wie eine Modem-, ISDN- bzw. DSL- (Digital Subscriber Loop) oder Satellitenverbindung durch Emulation nachzubilden.

Zur Erfüllung des Aspekts der umfassenden Emulationsumgebung werden auch Emulationsverfahren für Netzwerke mit Mehrfachpunktverbindungen, d.h. für Netze mit gemeinsamem Medium, benötigt. Sehr häufig werden LANs nicht mehr ausschließlich als abgegrenzte private Netze betrieben, sondern

sind Teil des Internet. Während dies früher vor allem durch Anbindung von Forschungseinrichtungen und Firmen an das Internet der Fall war, steigt die Verbreitung von angebundenen LANs auch in Privathaushalten. Die höhere verfügbare Bandbreite beispielsweise über das vorhandene Telefonnetz durch DSL wird in zunehmendem Maße durch LAN-Anbindungen von mehreren Endgeräten gemeinsam genutzt.

Unter dem Gesichtspunkt der Häufigkeit über das Internet verbundener LANs zeigt sich die Notwendigkeit für die Emulation von Netzen mit gemeinsamem Medium. Damit wird es möglich, im Rahmen einer umfassenden Emulationsumgebung vollständige, real vorkommende Netztopologien nachzubilden. Für kabelgebundene Netze mit gemeinsamem Medium ist es Aufgabe dieser Diplomarbeit, Emulationsverfahren zu entwickeln.

Mit der Entwicklung von Emulationsverfahren für Mobilfunk- und Ad-Hoc-Netze beschäftigt sich die ebenfalls im NET-Projekt angesiedelte Diplomarbeit [Dud02].

1.2 Aufgabenstellung

Im Rahmen des NET-Projekts an der Abteilung Verteilte Systeme soll eine Erweiterung der bestehenden Emulationsverfahren um Verfahren für Netze mit gemeinsamem Medium entwickelt werden.

Eine Erweiterung ist notwendig, da das bisherige Emulationskonzept für solche Netze nicht ausreichend ist. Durch Vorhandensein eines gemeinsamen Mediums beeinflussen sich im Gegensatz zu exklusiven Punkt-zu-Punkt-Verbindungen die Aktivitäten der einzelnen Teilnehmer. Sowohl zentralisierte als auch verteilte Ansätze sind für neue Emulationskonzepte denkbar.

In dieser Diplomarbeit sollen verschiedenen Konzepte für die Emulation von Netzen mit gemeinsamem Medium erarbeitet werden. Eine systematische Evaluation der Konzepte führt zur Auswahl eines erfolgversprechenden Emulationskonzepts. Von diesem ist eine prototypische Implementierung zu realisieren und diese entsprechend durch Messungen auf reale Tauglichkeit zu überprüfen.

Kapitel 2

Grundlagen

Dieses Kapitel vermittelt die technischen Grundlagen, auf denen diese Arbeit basiert. Zunächst erfolgt eine Einordnung in den Kontext bereits durchgeführter, vergleichbarer Arbeiten sowie eine Abgrenzung gegenüber diesen. Im Anschluß folgt eine Übersicht über Netze mit gemeinsamem Medium, deren Eigenschaften und im weiteren betrachtete Medienzugriffsprotokolle. Abschließend wird die NET-Umgebung vorgestellt.

2.1 Emulation, existierende Ansätze

2.1.1 Simulation, Realumgebung, Emulation

In Abschnitt 1.1 wurde bereits kurz der Entwicklungsprozeß neuer verteilter Anwendungen bzw. neuer Netzwerkprotokolle angesprochen. Er beginnt zumeist mit der mathematischen Analyse eines neuen Konzepts. Dies bedeutet aber, fast immer eine Abstraktion vom realen Modell vornehmen zu müssen, um das mathematische Modell als Abbild der Realwelt handhabbar zu halten. Insofern lassen sich Abweichungen durch Ungenauigkeiten, die aus der Abstraktion resultieren, kaum vermeiden. Es kann also nicht unmittelbar aus den Analyseergebnissen auf das spätere Realverhalten geschlossen werden. Für Analysen, die Aufgrund ihrer Komplexität nicht mehr sinnvoll mit Hilfe eines mathematischen Modells durchführbar sind, bietet sich die Simulation des neuen Konzepts an. Im Gegensatz zur mathematischen Analyse erlaubt die Simulation eine Analyse weitaus größerer und komplizierterer Betriebsszenarien. Naturgemäß ist auch eine Simulation nur wiederum durch eine Abstraktion vom tatsächlichen Verhalten realisierbar.

Da bislang stets Abstraktionen das Analyseergebnis zu verfälschen drohten, wird im Laufe der Entwicklung eine Implementierung des Konzepts notwendig, um mit dessen Hilfe Messungen des Realverhaltens vornehmen zu können. Hierzu wird eine Netzumgebung benötigt, die spezifische Auswirkungen auf das zu messende Objekt hat. Dies kann sich als schwierig bis unmöglich erweisen, sobald eine entsprechend große Meßumgebung verlangt wird. Diese würde, sofern überhaupt realisierbar, zu teuer und groß im Sinne von Platz- und anderem Ressourcenverbrauch werden. Gar völlig unmöglich ist eine reale Meßumgebung beispielsweise für einen Netztyp, für den noch kein verfügbares Zubehör existiert, da er sich z.B. gerade in der Entwicklung befindet. Trotzdem ist es dann sinnvoll, bereits verwertbare Informationen zu Wechselwirkungen zwischen Rechnernetz und Anwendung bzw. Protokoll zu gewinnen. Dies kann dann sowohl der Entwicklung der Software als auch des Netztyps selbst zu Gute kommen. Beispielsweise könnte bei einem neuen Mobilfunkstandard wie UMTS (Universal Mobile Telecommunication System) bereits Anwendungs-Software zur Videoübertragung parallel zu den dafür notwendigen tieferliegenden Netzschichten entwickelt und getestet werden, ohne dafür bereits funktionsfähige

UMTS-Netze zur Verfügung zu haben.

Eine Alternative zu den bislang vorgestellten Untersuchungsmethoden ist die Emulation. Sie vereint Vorteile beider Methoden in einem Kompromiß zwischen Simulation und Messung in Realumgebungen. Anhand einer Auswahl relevanter Kriterien, die eine Analyse verteilter Anwendungen bzw. Rechnernetzprotokolle betreffen, werden in Tab. 2.1 die Vor- und Nachteile von Simulation, Emulation und Realumgebung gegenübergestellt.

Kriterien	**Simulation**	**Emulation**	**Realumgebung**
Reproduzier-, Steuerbarkeit	+ gut möglich	+ gut möglich	− schwer möglich
Netzverkehr / Last	− künstlich	+ real	+ real
Vereinfachung ggü. Realwelt	− hohe Abstraktion	+ hoher Realismus mögl.	+ keine Abstraktion
Umgebung aufsetzen	+ einfach	+ einfach	− aufwendig
Skalierbarkeit	+ hoch	○ verschieden	− gering
Preis	+ nicht teuer	○ verschieden	− u.U. sehr teuer
Experimentdauer	− lang	+ Realzeit	+ Realzeit

Tabelle 2.1: Vor- und Nachteile von Simulation, Emulation und Realumgebung

Es zeigt sich deutlich, daß die Emulation eine vielversprechende Alternative zu Realumgebungen und eine Ergänzung zur Simulation darstellt.

2.1.2 Existierende Emulationsansätze

Zur Emulation von Netzverbindungen existiert bereits eine Anzahl von Arbeiten, die sich mit der Modellierung einer solchen Verbindung auf verschiedene Art und Weise beschäftigen. Jedoch fällt beim Betrachten der meisten bisherigen Arbeiten auf, daß diese sich oft ausschließlich auf Punkt-zu-Punkt-Verbindungen beschränken. Im folgenden werden die jeweiligen Arbeiten kurz charakterisiert.

[ZB93] bespricht zwei verschiedene Ansätze für lokale Versuchsanordnungen zur Messung verteilter Anwendungen. Zum einen kann in verteilter Weise auf verschiedenen Netzschichten mit Hilfe bereits vorhandener Mechanismen wie dem Echo-Dienst bei TCP bzw. UDP oder Source Routing bei IP der Netzverkehr einer Richtung gezwungen werden, den tatsächlichen Weg durch das Internet zu nehmen, ohne dabei Zugang zum entfernten Kommunikationssystem besitzen zu müssen. Die Kommunikation in umgekehrter Richtung nimmt einen direkten Weg über ein LAN, in dem z.B. eine verteilte Anwendung auf Client- und Server-Rechner abläuft. Da hierdurch stets nur in einer Kommunikationsrichtung der erwünschte Leitweg verfolgt wird, wird das Experiment in zwei Phasen ausgeführt und nach der ersten Phase die jeweils andere Richtung durch das Internet geleitet. Unter Vernachlässigung der Verzögerungen im Emulations-LAN läßt sich so aus den Ergebnissen beider Phasen auf das Gesamtverhalten schließen. Der zweite, zentralisierte Emulationsansatz beruht auf der Auswertung von Stichprobenmessungen bezüglich Paketverzögerung und deren Anwendung auf die Emulation (trace-driven) möglichst in Echtzeit. Da die Messungen von beispielsweise Round Trip Times selbst eine gewisse Zeit in Anspruch nehmen, sind die Meßwerte unter Umständen noch nicht verfügbar, wenn sie für die Emulation bereits benötigt werden. Durch Verfahren aus der Wahrscheinlichkeitstheorie werden deshalb möglichst genaue Meßwerte bei Bedarf vorausgesagt. In beiden Ansätzen erfolgt die Manipulation von Paketen durch Socket-Bibliotheksaufrufe, die durch stellvertretende Emulationsimplementierungen ersetzt werden.

Eine Emulation basierend auf der Auswertung von Stichprobenmessungen bezüglich Paketverzögerung sowie -verlust und deren Anwendung auf die Emulation (trace-driven) möglichst in Echtzeit sieht

[YR98] vor. Die Vorgehensweise erlaubt die Emulation auf einem einzigen Emulationsrechner ohne Zugriff zu externen Kommunikationspartnern. Aus den gemessenen ICMP Round Trip Times werden One Trip Times geschätzt und relativer Versatz sowie Drift der Rechneruhren durch Berechnungen entfernt. Außerdem wird die, aus der Zeitgranularität der Rechneruhren resultierende, Rundung der ICMP-Zeitstempel ausgeglichen. Die Implementierung ist auf Kernel-Ebene angesiedelt und unterstützt die Manipulation von TCP- und UDP-Paketen.

[IP94, Riz97, ACO97, NIS97, HR02] emulieren eine Netzverbindung allesamt in verteilter Weise durch Einführung von Verzögerung. Sie unterscheiden sich vor allem in der gewählten Implementierungsschicht im Betriebssystem. Delayline [IP94] ersetzt Socket-Bibliotheksaufrufe, Dummynet [Riz97] setzt am Übergang zwischen TCP und IP an, NIST Net [NIS97] arbeitet auf IP-Ebene und NET [HR02] emuliert auf Ebene der Sicherungsschicht. ONE [ACO97] wird nicht direkt auf einem Endgerät am Netz betrieben, sondern bildet einen Router mit zwei Netzverbindungen nach, indem Pakete von einem Netzanschluß auf Vermittlungsschicht abgefangen und an den anderen Netzanschluß weitergereicht werden. Zusätzlich zur Verzögerung bieten [IP94, NIS97, HR02] und neuere Versionen von [Riz97] auch die Nachbildung von Paket- bzw. Rahmenverlust. [NIS97] erlaubt darüber hinaus noch das Emulieren von umsortierten Paketen und Duplikaten. [HR02] geht in seiner Funktion insofern über die anderen Ansätze hinaus, als daß auch ein dynamisches Emulationsszenario unterstützt wird.

[Fal99] beschreibt die Erweiterung des zentralisierten Netzsimulators ns2 zu einem Emulator, der echten Netzverkehr verarbeiten kann. Hierzu wird der ereignisgesteuerte Simulator mit einem Echtzeit-Scheduling versehen. Die Verbindung des Simulatorkerns mit dem realen Netz wird über Standardbibliotheksschnittstellen zum Abfangen und Wiedereinspeisen von Paketen bzw. Rahmen hergestellt. Somit lassen sich durch den programmierbaren Simulatorkern praktisch beliebige Manipulationen am Netzverkehr vornehmen. Eingeschränkt wird dies jedoch durch die Verarbeitungsgeschwindigkeit der gesamten Emulations-Software, die durch ihre teilweise Skriptsteuerung und Implementierung im Benutzermodus entsprechend langsamer als direkte Emulationsansätze ohne Umweg über einen Simulator ist. Außerdem bringt die Simulatorschnittstelle zum realen Netz durch ihre Architektur unerwünschte zusätzliche Verzögerungen mit sich.

Einen ebenfalls zentralisierten Ansatz verfolgt [KGM+01]. Die beschriebene Emulationsumgebung Seawind besteht aus einer Reihe von Komponenten. Der zentrale Emulationsrechner beherbergt für jeden an der Emulation teilnehmenden Rechnerknoten einen Simulationsprozeß. Teilnehmende Rechner leiten ihren Netzverkehr jeweils durch einen solchen Simulationsprozeß bzw. empfangen von diesem. Zum Abfangen und Wiedereinspeisen von Netzverkehr existieren sogenannte Netzprotokolladapter. Zusätzlich können auf den teilnehmenden Rechnern Netzlastgeneratoren ausgeführt werden. Simulationsprozeß, Netzprotokolladapter und Netzlastgenerator kommunizieren über TCP-Verbindungen mit einem Steuerungsrechner mit graphischer Oberfläche zum Zwecke der Konfiguration sowie dem Mitschreiben von Emulationsprotokollen.

Weitergehende Funktionalität, die über die Emulation von Punkt-zu-Punkt-Verbindungen hinausgeht, bietet beispielsweise [DBCF95]. Für Funknetze mit einfachem, gemeinsamen Medium wird die Erkennung von Kollisionen unterstützt. Realisiert ist dies durch einen zentralisierten Emulationsknoten, der Kollisionen eingehender Rahmen erkennt und diese dann verwirft. Eine weitergehende Kollisionsbehandlung ist nicht vorgesehen. Neben Kollisionen emuliert der zentrale Knoten Paketverzögerung. Manipuliert werden UDP-Pakete durch Austausch der Socket-Bibliotheksaufrufe bzw. bei verteilten Anwendungen, die auf der verteilten Systemplattform ANSAware basieren, ohne weitere Änderungen.

In [ZL02] wird eine mögliche Emulationsumgebung bestehend aus einer Menge von UML-Knoten (User Mode Linux)[1] beschrieben. Dabei erfolgt die Kommunikation zwischen diesen UML-Knoten

[1] UML ist eine zusätzliche Zielsystemplattform-Architektur für Linux, die es erlaubt, einen UML-Kernel innerhalb eines Benutzerprozesses statt direkt auf der Hardware auszuführen.

über Multicast. Dies wird in der Arbeit effektiv als Emulation eines gemeinsamen Ethernet-Segmentes bezeichnet. Angesichts fehlender Limitierung auf eine gemeinsame Bandbreite sowie nicht nachgebildeter Kollisionen kann dies jedoch kaum realistisch sein. Allerdings liegt der Fokus der Arbeit nicht im Bereich der LANs, sondern der Mobile-Ad-Hoc-Netze. Die Emulation letzterer erfolgt mit Hilfe von dynamisch installierten Paketfiltern, die auf MAC-Adressen ansprechen. Dadurch werden Verbindungsunterbrechungen nachgebildet. Gesteuert werden die Filter durch Szenariobeschreibungen, die als ns2-Skript bzw. in einer davon abgeleiteten, vereinfachten Syntax vorgegeben werden können.

Aus der Betrachtung dieser Arbeiten läßt sich eine Klassifizierung der Emulationskonzepte ableiten. Die Konzepte lassen sich nach verschiedenen, zueinander orthogonal liegenden Merkmalen einordnen. Die jeweiligen Merkmalsausprägungen eines Emulationskonzepts können durchaus auch auf Stufen zwischen den direkt angegebenen Klassifikationen liegen. Die Literaturangaben hinter den folgenden Merkmalsausprägungen dienen zur Einordnung bereits erwähnter Emulationskonzepte.

- Implementierungsschicht

  Die Schicht, auf der ein Konzept implementiert wird, beeinflußt verschiedene Aspekte wie Umsetzbarkeit, Echtzeitfähigkeit oder Genauigkeit der Emulation.

  - Benutzermodus
    - proprietäre Bibliothek [DBCF95]
    - Ersatz für Standardbibliothek (z.B. Socket-Aufrufe in libc) [ZB93, IP94, ZL02]
    - Abfangen und Wiedereinspeisen von Netzverkehr über bestimmte Systemschnittstellen [ACO97, Fal99]
  - hybrid, Benutzer- und Systemmodus (z.B. Modellberechnung im Benutzermodus und davon abhängig Steuerung von Systemparametern) [HR02]
  - Systemmodus
    - Ersatz für Systemaufrufe
    - Transportschicht [Riz97, YR98]
    - Vermittlungsschicht [NIS97]
    - Sicherungsschicht $\hat{=}$ Netzwerkgerätetreiber und stellt somit die unterste Software-Schicht über der Hardware dar. [HR02]

- Architektur

  Es existieren zwei grundlegende Vorgehensweisen zur Umsetzung des Emulationsproblems: Eine verteilte oder eine zentralisierte Lösung, wobei eine hybride Mischform ebenfalls möglich ist.

  Die Architektur eines Emulationskonzepts zählt zu einer der folgenden Klassen:

  - verteilt [ZB93, IP94, ACO97, Riz97, YR98, NIS97]
  - hybrid, d.h. verteilt und zentral [HR02, ZL02]
  - zentral [ZB93, DBCF95, Fal99, KGM+01]

- Emulationsgrad

  Es ist schwierig, für den Emulationsgrad eine sinnvolle Maßeinheit festzulegen. Zwar besitzt eine Realimplementierung sicherlich keine Abstraktion, doch ist das andere Extrem, nämlich die komplette Abstraktion, schwierig zu fassen. Im folgenden wird eine Auswahl verschiedener Emulationsgrade, sortiert von hoher zur geringsten Abstraktion, betrachtet.

- abstraktes Berechnungsmodell (z.B. aus der Warteschlangentheorie)
- trace-driven (Messungen aus realer Umgebung steuern Parameter während Emulationsablauf) [ZB93, YR98]
- Emulation einer bestimmten Netzschicht durch Einführung von Bandbreitenlimitierung, Verzögerung und evtl. Bitfehler oder verlorene Pakete [ZB93, DBCF95, IP94, ACO97, Riz97, NIS97, Fal99, KGM+01, HR02, ZL02]
- originalgetreue Nachbildung einer bestimmten Netzschnittstelle in Software (z.B. spezieller Gerätetreiber auf Sicherungsschicht, der ein Medienzugriffsverfahren emuliert und auf der Emulationsumgebung als Transportmedium aufsetzt)

Abb. 2.1 veranschaulicht die drei identifizierten Klassifizierungsachsen mit ihren jeweiligen Merkmalsausprägungen auf graphische Weise.

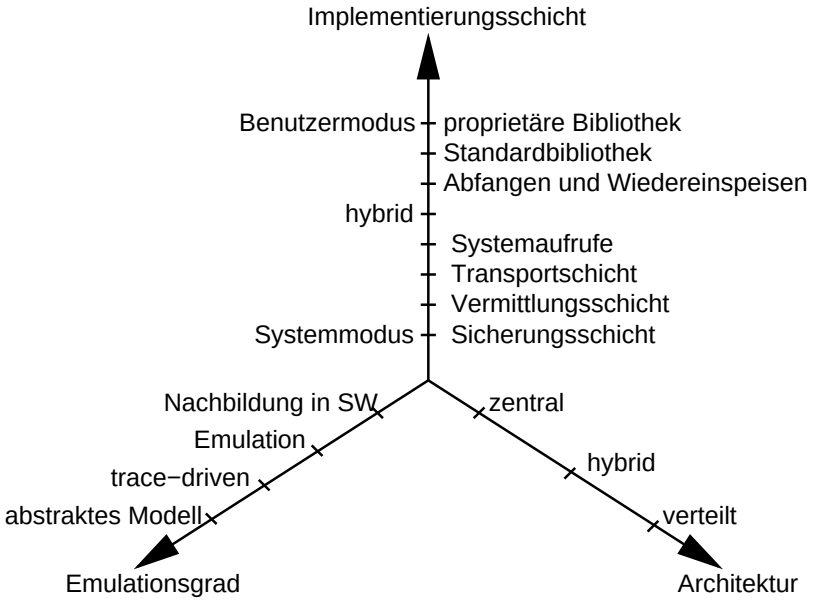


Abbildung 2.1: Netz-Emulationskonzepte und ihre Merkmalsausprägungen

Zur Emulation kabelgebundener Netze mit gemeinsamem Medium wie beispielsweise LANs existiert bislang keine Untersuchung.

2.2 Netze mit gemeinsamem Medium

Die Gemeinsamkeit des Mediums solcher Netze bringt die Notwendigkeit des exklusiven Zugriffes auf das Medium mit sich. Andernfalls wäre durch die gegenseitige Beeinflussung von Signalen unterschiedlicher Stationen eine fehlerfreie Übertragung von Nachrichten nicht zu gewährleisten. Der Exklusivzugriff kann durch unterschiedliche Zugangssteuerungsverfahren gewährleistet werden.

Im frühen Entwicklungsstadium von Netzen mit gemeinsamem Medium wurden statische Kanalaufteilungsverfahren eingesetzt. Hierbei wird beispielsweise beim Zeitmultiplexverfahren jeder Station am Netz ein bestimmter Zeitschlitz zugeordnet, während dem sie auf dem Medium senden darf. Die

Zeitschlitze aller Stationen werden periodisch nacheinander zugeteilt. Dadurch entsteht eine feste Kanalzuteilung. Kollisionen oder anders geartete gegenseitige Beeinflussungen der Stationen durch ihre Aktivitäten sind ausgeschlossen. Der zugeteilte Kanal kann je nach Aufteilungsverfahren z.B. ein Frequenzband bei Frequenzmultiplex, ein Zeitschlitz bei Zeitmultiplex oder ein Kodierschema bei Kodemultiplex sein. Mit Hilfe bekannter Gesetze aus der Systemtheorie läßt sich aus der Kanalbandbreite die tatsächlich erreichbare Übertragungsrate ermitteln. Hierzu werden die Erkenntnisse von Nyquist, 1924 für rauschfreie Kanäle und vor allem Shannon, 1948 für Kanäle mit thermischem Rauschen, was selbstverständlich wesentlich realistischer ist, verwendet. Eine Emulation eines solchen Netzes mit statischen Kanalaufteilungsverfahren läßt sich also bereits mit dem vorhandenen Emulationswerkzeug NETShaper realisieren, da die Limitierung der Sendebandbreite jeder einzelnen Station den zugeteilten Kanal ausreichend beschreibt.

In Computernetzen entsteht häufig wechselnder Verkehr mit Bursts. Für dieses Verkehrsmuster sind statische Verfahren jedoch ungeeignet. Bandbreite ungenutzter Kanäle geht unweigerlich verloren und kann nicht anderweitig verwendet werden. Hier würde man sich wünschen, die Gesamtbandbreite des gemeinsamen Mediums dynamisch je nach Bedarf an die Stationen verteilen zu können. Zu diesem Zweck wurden dynamische Kanalzuteilungsverfahren entwickelt, die im folgenden weiter betrachtet werden.

Abb. 2.2 zeigt die bereits diskutierte Unterteilung von Zugangssteuerungsverfahren in die Klasse der statischen Aufteilungs- und der dynamischen Zuteilungsverfahren.

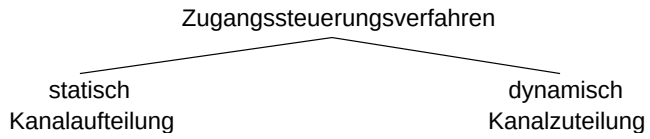


Abbildung 2.2: Klassen von Zugangssteuerungsverfahren

2.2.1 Eigenschaften dynamischer Kanalzuteilungsverfahren

In Anlehnung an [Tan96] weisen Kanalzuteilungsverfahren für Netze mit gemeinsamem Medium die folgenden grundlegenden Eigenschaften auf:

1. Annahme eines einzigen Übertragungskanals: Es gibt keinen externen Signalisierungskanal. Auf dem gemeinsamen Medium können alle angeschlossenen Stationen gleichberechtigt sendend und empfangend auf die Hardware zugreifen.

2. Kollisionen: Kollisionen entstehen durch zeitliche Überlappung von übertragenen Rahmen und können von allen Stationen erkannt werden. Es gibt keine anderen Fehler als Kollisionen.

3. Zeit

   - kontinuierlich
   - diskrete Intervalle, sog. Zeitschlitze

4. Träger

   - mit Trägererkennung: Solange ein Träger auf dem Kanal erkannt wird, erfolgt erst gar kein Sendeversuch.

- ohne Trägererkennung: Stationen beginnen bei Bedarf zu senden. Erst im Anschluß kann festgestellt werden, ob die Übertragung erfolgreich war oder nicht.

5. geteilte Bandbreite: Zu allen Zeitpunkten t und für alle Stationen $r \in R$ am gemeinsamen Medium R gilt für die momentanen Übertragungsraten $\dot{b}(t)$: $\sum_{r \in R} \dot{b}_r(t) \leq \dot{b}_R(t)$, d.h. die Summe der Raten aller Stationen ist stets kleiner oder gleich der maximalen Rate auf dem gemeinsamen Medium.

Darüber hinausgehend lassen sich weitere Eigenschaften zur Unterscheidung verschiedener Verfahren feststellen. Die Gesamtheit der Eigenschaften soll im Hinblick auf das Erarbeiten von Emulationskonzepten in Kapitel 4 ein vernünftiges Erforschen des Parameterraumes unterstützen. Ein solches Vorgehen ist zusammen mit der Suche nach Invarianten hilfreich bei der gezielten Entwicklung von Simulationen [PF97], läßt sich sicherlich aber in diesem Fall ebenso auf Emulationen anwenden. Die folgende Aufzählung belegt keineswegs den Anspruch auf absolute Vollständigkeit.

- Das Verhalten des Netzes ist abhängig von der Netzlast, z.B. ist die verfügbare Bandbreite oder auch die Verzögerung bis zum Medienzugriff von der Last abhängig.
- Fairneß in Form einer Sendegarantie z.B. bei Token Ring oder Token Bus.
- Ausschluß des Verhungerns von Stationen, d.h. nicht nur Fairness, sondern auch eine maximale Wartezeit bis zur nächsten Sendeerlaubnis.
- Die Signalausbreitungszeit zwischen Stationen ist von der Netztopologie abhängig.

2.2.2 Mehrfachzugriffsprotokolle

Dynamische Kanalzuteilungsverfahren, welche die Zugriffsreihenfolge auf ein gemeinsames Medium bestimmen, werden auch Mehrfachzugriffsprotokolle genannt. Eine Auswahl solcher Protokolle und ihre hierarchische Einordnung in die Gesamtheit der Zugangssteuerungsverfahren ist in Abb. 2.3 dargestellt.

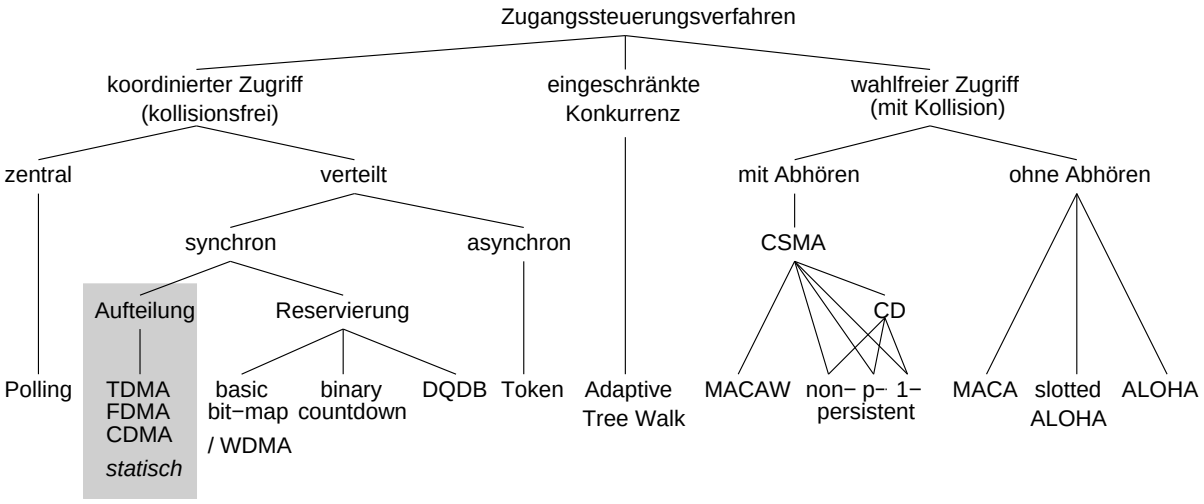


Abbildung 2.3: Zugangssteuerungsverfahren

Die grau unterlegten, statischen Kanalaufteilungsverfahren werden, wie bereits in Abschnitt 2.2 erwähnt, nicht weiter betrachtet. Es würde den Rahmen dieser Arbeit sprengen, sämtliche dargestellten

Protokolle zu beschreiben und zu vergleichen. Hierzu sei an dieser Stelle auf Standardliteratur wie beispielsweise [Tan96] verwiesen. Die Darstellung soll in Verbindung mit Abschnitt 2.2.1 vielmehr für einen Überblick über die Gemeinsamkeiten und Unterschiede der Verfahren hilfreich sein. Als Blätter des Hierarchiebaums sind jeweils die verschiedenen Mehrfachzugriffsprotokolle abgebildet. Im Vergleich zu Abb. 2.2 erfolgt nun die Unterteilung nicht primär nach statischen oder dynamischen Verfahren. Stattdessen drückt die Anordnung der Blätter von links nach rechts eine zunehmende Zahl möglicher Kollisionen aus, wobei der gesamte linke Unterbaum mit der Wurzel „koordinierter Zugriff“ Verfahren beherbergt, die Kollisionen vollständig vermeiden. Verfahren mit eingeschränkter Konkurrenz gehen einen Kompromiß zwischen koordiniertem und wahlfreiem Zugriff ein, um die Zahl möglicher Kollisionen möglichst gering zu halten und gleichzeitig Vorteile wie beispielsweise kurze Verzögerung beim Medienzugriff zu erreichen. Es ist anzumerken, daß neben den explizit als verteilt markierten Verfahren im rechten Unterbaum des Knotens „koordinierter Zugriff“ sämtliche Verfahren mit eingeschränkter Konkurrenz oder wahlfreiem Zugriff eine verteilte Architektur besitzen.

2.3 NET-Umgebung

Die Emulationsumgebung des NET-Projekts befindet sich noch im Aufbau. Sie besteht auf Hardware-Seite aus einem Cluster mit 64 Knoten. Die Knoten sind handelsübliche PCs mit Linux als Betriebssystem. Sie sind über einen Gigabit-Ethernet-Switch verbunden. Für administrative Zwecke existiert für jeden Knoten zusätzlich ein zweite Verbindung über einen Fast-Ethernet-Switch. Die NET-Hardware und ihre Verkabelung zeigt Abb. 2.4.

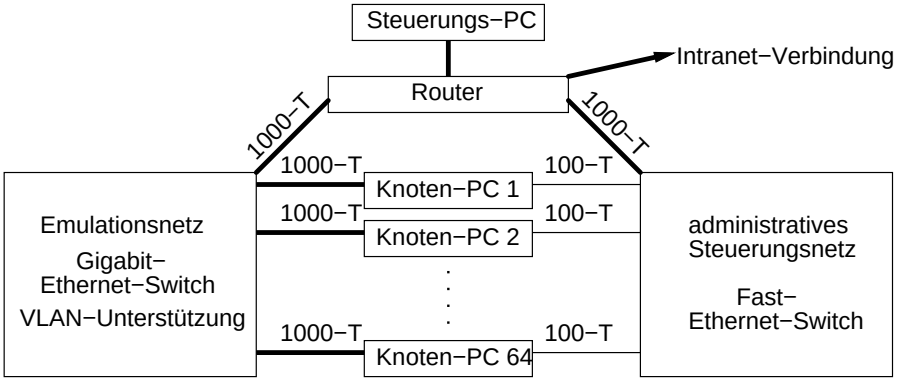


Abbildung 2.4: Die NET-Hardware

Der Gigabit-Switch unterstützt VLANs (virtual local area networks) nach IEEE 802.1Q. Diese Eigenschaft wird zum Aufbau virtueller Netztopologien eingesetzt, ohne Änderungen an der physikalischen Verkabelung vornehmen zu müssen. Linux bietet für jede VLAN-Gruppe mit einer eindeutigen Markierung („tag“), in der ein Knoten Mitglied ist, jeweils ein virtuelles Netzwerkgerät, das sich in der Verwendung nicht von einem „normalen“ Netzwerkgerät unterscheidet. Insofern ist der Einsatz von VLANs für die Netzschichten über der Sicherungsschicht und somit letztendlich auch für Anwendungen vollkommen transparent. Der Effekt des Einsatzes markierter („tagged“) VLANs ist die Ausnutzung einer physikalischen Kabelverbindung für mehrere unabhängige virtuelle Netzanschlüsse.

Abb. 2.5 veranschaulicht anhand eines einfachen Topologiebeispiels den Einsatz von VLANs. VLANs mit nur zwei Knoten als Teilnehmer entsprechen einer einfachen Punkt-zu-Punkt Verbindung. Beispiele

hierfür sind die VLANs 2 bis 5. VLANs mit mehr als zwei Knoten sind nichts weiter als ein abgetrenntes, privates Netzsegment. Ein solches Segment ist im Beispiel durch VLAN 1 repräsentiert.

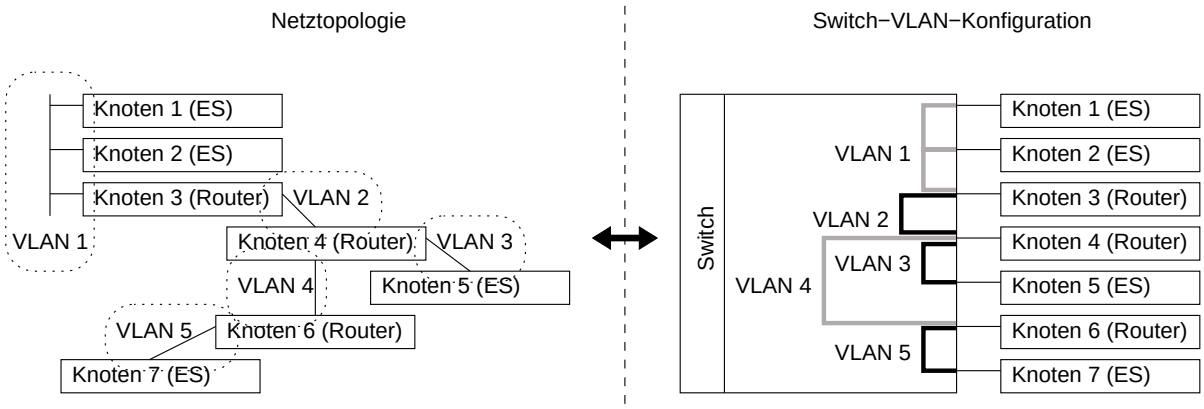


Abbildung 2.5: Beispiel zur Konfiguration von Topologien durch VLANs

Eine berechtigte Frage ist, warum denn beispielsweise zur Emulation von Ethernet-LANs nicht einfach die meist abwärtskompatiblen Netzwerkkarten auf die gewünschte niedrigere Übertragungsrate gezwungen werden. Im Prinzip müßte dies effektiv einem realen Ethernet-Segment entsprechen. Für den Fall, daß eine Station nur ein VLAN-Gerät exklusiv auf dem Gigabit-Netzadapter eingerichtet hat, d.h. es existiert nur eine virtuelle pro physikalischer Verbindung, würde dies funktionieren, sofern es akzeptabel ist, daß es sich durch den Switch dann immer noch um kollisionsfreies Ethernet handelt. Sobald die physikalische Verbindung einer Station zum Switch durch mehrere VLANs in mehr als eine virtuelle Verbindung aufgeteilt wird, schlägt jedoch dieser einfache Emulationsansatz fehl. Die eingestellte Übertragungsrate würde dann für sämtliche virtuelle Verbindungen gemeinsam gelten und ist nicht, wie eigentlich notwendig, spezifisch pro Verbindung regelbar.

Auf Software-Seite existiert in der NET-Umgebung bislang das weiter oben bereits angesprochene Emulationswerkzeug NETShaper sowie eine Unterstützung für dynamische Emulationsparameteränderungen.

NETShaper wird konfiguriert durch Parameter für eine maximale Übertragungsrate, für Signalausbreitungsverzögerung und für eine Rahmenverlustrate. Anhand dieser Parameter bildet das Emulationswerkzeug eine Punkt-zu-Punkt-Verbindung auf Sicherungsschicht nach, indem jeder zu sendende Rahmen mit einer entsprechenden Verzögerung, die auch eine Warteschlangenbehandlung beinhaltet, beaufschlagt wird. Zusätzlich liefert NETShaper bei Netzsättigung eine Rückmeldung an die Vermittlungsschicht, statt Rahmen einfach zu verwerfen. Somit wird der ordnungsgemäße Betrieb von Leitwegverfahren wie beispielsweise Fair Queueing erlaubt.

Die dynamische Änderung von Emulationsparametern wird durch Konfigurations-Dienstprozesse auf den Emulationsknoten unterstützt. Diese Dienstprozesse empfangen über UDP-Pakete Änderungswünsche von einem zentralen Steuerungsknoten und konfigurieren daraufhin das Emulationswerkzeug auf dem jeweiligen Knoten.

Kapitel 3

Evaluationskriterien

Die später in Kapitel 4 vorgestellten Emulationskonzepte sollen systematisch evaluiert werden. Zu diesem Zweck erscheint es sinnvoll, bereits vor dem Erarbeiten der Konzepte, Kriterien für eine anschließende Evaluation aufzustellen. Auf diese Weise ist eine sinnvolle Bewertung möglichst unabhängig von spezifischen Konzepteigenschaften möglich. Im Anschluß werden die einzelnen Kriterien vorgestellt. Für jedes Kriterium folgt eine Motivation sowie die Festlegung eines Bewertungsmaßes und dessen Einordnung in Bezug auf die Bewertung.

1) **Anzahl emulierter Netztypen.** Im vorangegangenen Kapitel 2 wurde bereits aufgezeigt wie sich Netze mit gemeinsamem Medium durch den notwendigen Exklusivzugriff auf das Medium grundlegend durch ihre verschiedenen Medienzugriffsverfahren unterscheiden. Unter der Annahme, daß sich diese verschiedenen Verfahren durch Parametrisierung eines allgemeinen Modells emulieren lassen, ergibt sich das erste wichtige Kriterium, nämlich die Anzahl emulierbarer Netztypen.

Das Maß für die Anzahl emulierter Netztypen ist eine natürliche Zahl größer Null.

Ein Idealkonzept sollte allein durch Parametrisierung möglichst sämtliche Netze mit gemeinsamem Medium emulieren können. Je höher also die Anzahl in diesem Kriterium, desto besser – da allgemeiner verwendbar – ist das Konzept.

2) **Realismus.** Das Abstraktionsniveau eines Konzepts beeinflußt die Genauigkeit bzw. den Realismus der Emulation im Vergleich zum Betrieb eines realen Netzes. Die höchste Genauigkeit erhält man, indem man auf jegliche Abstraktion verzichtet und eine Realimplementierung einsetzt. Ausgehend von dieser „Null-Abstraktion“ kann sich das Emulationsverhalten umso mehr vom Realverhalten entfernen, je höher die Abstraktion gewählt wird.

Eine sinnvolle, absolute Maßeinheit für den Realismus zu finden, gestaltet sich schwierig. Ich schlage eine Skala in Prozent vor, die anzeigt wie viele der Spezifika verschiedener Netztypen durch ein Konzept mit akzeptabler Abweichung vom Realverhalten emuliert werden können. Der Maximalwert beträgt 100 % für ein, mit vernachlässigbaren Abweichungen, reales Emulationsverhalten.

Je höher der Prozentwert, desto besser – da näher am Realverhalten – ist ein Konzept.

3) **Single Point of Failure.** Um eine hohe Ausfallsicherheit zu gewährleisten, dürfen keine Single Points of Failure im System vorhanden sein. Diese sind evtl. durch Redundanz behebbar.

Die Anzahl der Single Points of Failure ist eine natürliche Zahl, die im Intervall $(0, N+X)$ liegt. N steht für die Anzahl der Knoten, die an der Emulation teilhaben. X ist die Anzahl evtl. möglicher zusätzlicher Knoten, die zur Emulationssteuerung notwendig sind. Beispielsweise gilt $X = 1$ in einem zentralisierten System mit einem Steuerungsknoten.

Je niedriger die Anzahl der Single Points of Failure, desto besser ist ein Konzept. Sicherlich ist dabei ein Unterschied wie zwischen $N+X-y$ und $N+X-y-1$ nicht als tatsächlich nutzbringende

Verbesserung zu interpretieren. Hauptsächlich geht es um die Klassen mit 0, X, N und $N+X$ Single Points of Failure im System.

4) **Flaschenhals.** Um eine hohe Skalierbarkeit zu gewährleisten, dürfen keine Flaschenhälse im System vorhanden sein. Diese sind evtl. durch Replikation behebbar.

Für die Anzahl der Flaschenhälse gilt das Intervall aus dem vorangehenden Kriterium 3.

Die Bewertung erfolgt ebenfalls analog.

5) **Overhead.** Verschiedene Lösungsansätze unterscheiden sich im verursachten Kommunikationsaufwand. Da bei der Netzemulation die Kommunikation zur Steuerung des Emulationsverfahrens zusätzlich zum primären Netzverkehr des emulierten Systems mit seinen verkehrserzeugenden Anwendungen auftritt, lassen sich Verfahren an Hand ihres Kommunikations-Overheads vergleichen. Sowohl der Overhead in der Steuerkommunikation als auch allgemein der Overhead, der durch ein Verfahren – z.B. durch interne Modellberechnungen – erzeugt wird, ist ein wichtiges Kriterium.

Das Maß für den Overhead ist der Prozentanteil zusätzlich generierter Rechnerlast durch das Emulationsverfahren gegenüber der Rechnerlast, die in einer realen Umgebung verursacht werden würde. Die zusätzliche Last ist in ihrem Wert von der Systemlast abhängig. Von niedrigen bis möglichst hohen Systemlasten sollte der Overhead keine negativen Auswirkungen durch Beeinflussung der zu testenden Software nach sich ziehen. Wichtig ist das Verhalten bei hoher Systemlast, denn natürlich möchte man möglichst die volle Leistungsfähigkeit der Emulationsumgebung ausnutzen. Damit nun durch die Zusatzlast möglichst wenig zusätzliche Kommunikation bzw. Verzögerung auftritt und die Meßergebnisse verfälschen kann, muß der Overhead minimiert werden [HR02].

Ein Konzept ist also besser, je weniger Overhead es erzeugt. Das wohl kaum erreichbare Optimum liegt bei 0 %.

6) **Echtzeitfähigkeit.** Um nicht nur geringe sondern die – vor allem bei LANs und MANs verwendeten – hohen Bandbreiten emulieren zu können, muß der Overhead, der in der Emulation unerwünschte Verzögerungen einführt, möglichst gering gehalten werden. Auf diese Weise kann eine Echtzeitfähigkeit des Emulationssystems überhaupt nur erreicht werden. Es wird hier nicht der Anspruch auf harte Echtzeitfähigkeit gestellt, da dies mit der zur Verfügung stehenden NET-Umgebung und dem dort verwendeten Standard-Linux-Kernel nicht umsetzbar wäre.

Als Maß für das Kriterium wird die durchschnittliche Verzögerung, die pro zu verarbeitendem Rahmen in der Emulation anfällt, gewählt.

Je geringer die Verzögerung, desto besser ist ein Konzept.

7) **Netzlast.** Ein geringer Overhead ist außerdem die Voraussetzung, um eine hohe Netzlast bei hohen Bandbreiten in der Emulation verarbeiten zu können.

Die emulierbare Last wird absolut in Bit/s angegeben. Sie wird u.a. von der tatsächlichen, physikalischen Bandbreite des Netzes der Emulationsumgebung beeinflußt. Um bei vergleichbaren absoluten Werten zu bleiben, wird eine physikalische Bandbreite von 1 GBit/s angenommen. Dies entspricht den technischen Daten der NET-Umgebung.

Je höher die emulierbare Netzlast, desto besser – da die Emulationsumgebung besser ausgenutzt werden kann – ist ein Konzept.

8) **Umsetzbarkeit.** Bislang unbetrachtet blieb die Eignung eines Emulationskonzepts für die Umsetzung in der gegebenen NET-Umgebung. Abgesehen von der Auswahl eines nach theoretischen Gesichtspunkten möglichst optimalen Konzepts ist die Eignung zur Umsetzung ein hochgradig entscheidendes Evaluationskriterium für die Auswahl eines erfolgversprechenden Konzepts, welches dann prototypisch implementiert werden soll.

Die Umsetzbarkeit eines Konzepts wird auf einer Ordinalskala von 1 bis 6 ausgedrückt. Dabei bedeutet 1, daß ein Konzept problemlos umsetzbar ist und 6, daß es nicht umsetzbar ist. Unter Berücksich-

tigung der NET-Umgebung werden die Werte mit folgenden Beschreibungen assoziiert:

1. Das Konzept kann unter Verwendung der vorhandenen Progammierschnittstellen im Linux-Kernel und unter Beschränkung auf Einprozessormaschinen der Pentium-Familie als Kernel-Modul implementiert werden und ist somit prinzipiell auf jedem derartigen Linux-Rechner ohne Änderungen lauffähig.

2. Das Konzept kann zwar als Kernel-Modul implementiert werden, kommt aber im Gegensatz zu 1. nicht mit den vorhandenen Programmierschnittstellen aus, sondern benötigt Veränderungen am Kernel mit Hilfe von Quelltext-Patches.

3. Das Konzept läßt sich gegenüber 2. nur vorwiegend bzw. ausschließlich als Kernel-Patch umsetzen und kann somit nicht auf einer Vielzahl von Linux-Rechnern dynamisch zur Laufzeit hinzugeladen und eingesetzt werden.

4. Das Konzept weist gegenüber 3. weitere Abhängigkeiten von fremden Software-Paketen auf. Beispielsweise wird ein anderes Kernel-Modul benötigt, das nicht zum Standard-Linux-Kernel zählt.

5. Das Konzept weist gegenüber den vorhergehenden Werten einen deutlich erhöhten Aufwand zur Umsetzung auf.

6. Es ist nicht mit vertretbarer Zeit und Aufwand möglich, das Konzept im Linux-Kernel für die NET-Umgebung umzusetzen. Beispielsweise würde ein anderes Betriebssystem mit unterschiedlichem Systemaufbau benötigt werden.

Je niedriger der Wert ist, desto besser ist ein Konzept bezüglich seiner Umsetzbarkeit einzuordnen.

9) **Relevanz.** Dieses Kriterium beschreibt die praktische Relevanz der emulierten Netztypen für die Emulation. Eine diesbezügliche Bewertung dient zur Unterscheidung verschiedener Netztypen. Es ist sinnvoll, häufig in der Praxis eingesetzte Typen solchen vorzuziehen, die nur sporadisch Verwendung finden und eine geringe Bedeutung haben.

Die Bewertung erfolgt auf einer Relativskala.

Dabei ist die Ethernet-Familie sicherlich mit Abstand der relevanteste Netztyp, um darauf Testläufe mit verteilter Software und Netzprotokollen durchzuführen.

Kapitel 4

Emulationskonzepte

Dieses Kapitel beschreibt verschiedene erarbeitete Emulationskonzepte für Netze mit gemeinsamem Medium. Zu Beginn werden die bereits in Abschnitt 2.1.2 identifizierten Einordnungskriterien in Bezug auf die Entwicklung von Konzepten diskutiert. Anschließend werden Emulationskonzepte verschiedener Ausprägungen der Einordnungskriterien vorgestellt. Das Kapitel schließt mit einer Übersicht über die vorgestellten Konzepte.

4.1 Einordnungskriterien

Basierend auf den bereits identifizierten Einordnungskriterien für Emulationskonzepte werden im folgenden die Auswirkungen verschiedener Merkmalsausprägungen betrachtet.

- Implementierungsschicht

  Die Schicht, auf der ein Konzept implementiert wird, beeinflußt verschiedene Aspekte wie Umsetzbarkeit, Echtzeitfähigkeit oder Genauigkeit der Emulation.

  Für die in diesem Kapitel vorgestellten Emulationskonzepte für Netze mit gemeinsamem Medium beschränkt sich die Auswahl auf solche, die auf Sicherungsschicht im Systemmodus implementiert werden. Diese Schicht ist die niedrigste, die ohne spezielle Hardware bedient werden kann und ermöglicht für reine Software-Lösungen den höchsten Grad an Realismus [HR02]. Aus diesem Grund werden die höheren Implementierungsschichten nicht näher betrachtet, sondern der bisherige Arbeitsbereich der NET-Umgebung beibehalten.

- Architektur

  Sicherlich wünscht man sich eine verteilte Lösung in der Hoffnung, damit hohe Ausfallsicherheit und Skalierbarkeit zu erreichen. Leider erfüllt ein verteilter Ansatz nicht automatisch diese Wünsche. Dies zeigt bereits ein Vergleich einfacher Verfahren zum gegenseitigen Ausschluß, einem wichtigen Synchronisationsproblem in verteilten Systemen. So kommt ein zentralisierter Algorithmus mit Koordinator mit einer kleinen konstanten Anzahl an Nachrichten für den Eintritt in den kritischen Bereich aus. Natürlich stellt der Koordinator sowohl einen Single Point of Failure als auch einen Flaschenhals dar. Der verteilte Algorithmus nach Lamport (1987) benötigt dagegen aber sogar eine Anzahl an Nachrichten, die linear von der Anzahl der Knoten im verteilten System abhängt. Außerdem stellt hier jeder einzelne Knoten einen Single Point of Failure dar, so daß sich insgesamt so viele Single Points of Failure wie Knoten im System befinden. In diesem Fall ist also die verteilte Variante in Bezug auf die Anzahl der Single Points of Failure und den

Kommunikationsaufwand der zentralisierten unterlegen. Vor diesem Hintergrund sollte man nicht voreilig zentralisierte Lösungen von vornherein ausschließen, sondern ebenso sorgfältig wie die verteilten auf ihre Eignung prüfen.

- Emulationsgrad

  Ein weitläufig parametrisierbares Konzept muß zwangsläufig auf einem abstrahierenden mathematischen Modell aufbauen. Es ist nicht möglich eine exakte Umsetzung des Realverhaltens verschiedener Netztypen herzustellen, ohne die unterschiedlichen Verhalten wenigstens teilweise durch eigene Algorithmen zu modellieren. Zwar ist nun die Auswahl aus verschiedenen implementierten Algorithmen mit Hilfe eines Parameters möglich, jedoch entspricht dies nicht der gewünschten allgemeinen Parametrisierbarkeit. Denn faktisch würde in einem solchen Fall nur die Implementierung geschickt ausgetauscht werden.

Ein bestimmtes Emulationskonzept besteht aus jeweils einer Merkmalsausprägung für jede der Klassifikationsachsen. Das heißt ein Konzept setzt sich aus einer Kombination von Merkmalen zusammen. Hierbei ist eine ganze Reihe sinnvoller Kombinationen möglich. Abb. 4.1 stellt eine Auswahl von Kombinationen auf graphische Weise dar. Die entsprechenden Konzepte werden in den folgenden Abschnitten beschrieben.

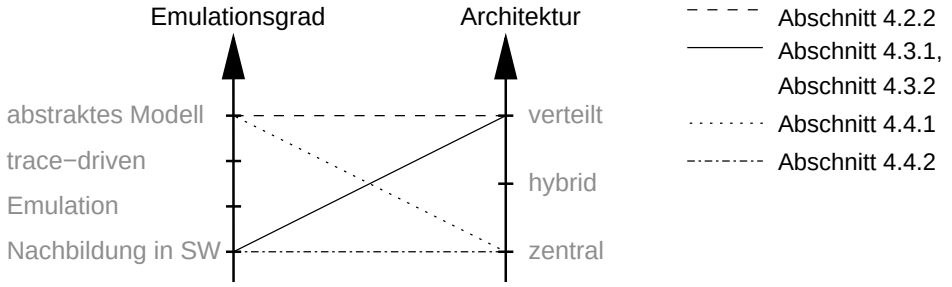


Abbildung 4.1: Auswahl von Emulationskonzepten und ihre Merkmalsausprägungen

Bei vielen der folgenden Konzepte wird von der Annahme ausgegangen, daß die physikalische Übermittlungszeit von Nachrichten in der Emulationsumgebung vernachlässigbar klein gegen die emulierten Verzögerungen ist. Soll beispielsweise in der NET-Umgebung mit einer physikalischen Übertragungsrate von 1 GBit/s ein Netz mit einer Rate von 10 MBit/s emuliert werden, so unterscheiden sich die Übermittlungszeiten um 2 Größenordnungen. Das heißt ein Rahmen, der zur Übertragung im emulierten Fall 1,2 Millisekunden benötigt, wird in der Emulationsumgebung in nur 12 Mikrosekunden übermittelt.

Diese Annahme begründet sich im Fehlen des Begriffs einer globalen Zeit in asynchronen verteilten Systemen. Durch die getätigte Vernachlässigung lassen sich bestimmte notwendige, feingranulare Zeitmessungen überhaupt erst vornehmen.

4.2 Hoher Emulationsgrad

Nach der Definition eines allgemeinen Emulationsmodells wird in diesem Abschnitt ein Konzept mit hohem Emulationsgrad erläutert. Bei dem Konzept handelt es sich um eine spezifische Implementierung des allgemeinen Modells.

4.2.1 Allgemeines Emulationsmodell

Ein allgemeines Modell zur Emulation läßt sich wie folgt definieren:

Definition 1 *Es existiert eine Menge $E = \{e_1, \ldots, e_m\}$ von Eingabeparametern e_i, deren Werte sich durch Messungen in der Emulationsumgebung erschließen. Außerdem existiert eine Menge $A = \{a_1, \ldots, a_n\}$ von Ausgabeparametern a_j, die direkt zur Steuerung der Emulation verwendet werden können. Die Abbildung von Ein- auf Ausgabeparameter erfolgt durch eine geeignete Relation $F : E \rightarrow A$.*

Für ein konkretes Modell gilt es, die passenden Ein- sowie Ausgabeparameter zu identifizieren und eine Berechnungsvorschrift für die Abbildung zu finden.

Die schematische Funktionsweise des allgemeinen Emulationsmodells ist in Abb. 4.2 dargestellt. Beispielsweise könnte die durch die Knoten generierte Netzlast gemessen werden und als Eingabeparameter dienen. Diese Last wird dann durch die Abbildung in Ausgabeparameter zur Steuerung der Emulation, wie z.B. den bei der aktuellen Last verfügbaren Durchsatz, umgewandelt. Die einzelnen Knoten wenden den Wert für den Durchsatz dann in Form von eingeführten Verzögerungen für zu sendene bzw. auszuliefernde Rahmen an.

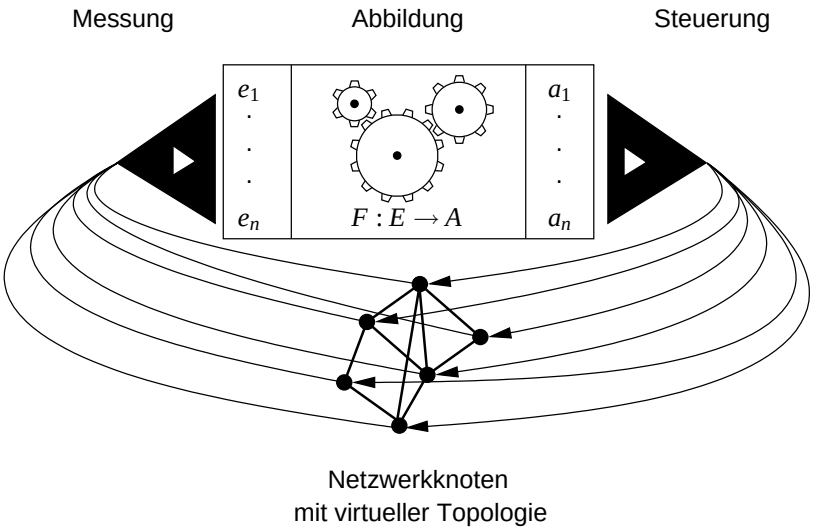


Abbildung 4.2: Schematische Funktionsweise des allgemeinen Emulationsmodells

4.2.2 Netzlast-Verhalten

Das im folgenden beschriebene Konzept läßt sich bezüglich der Implementierungsschicht im Systemmodus einordnen, da auf Sicherungsschicht Übertragungsratenlimitierung und Verzögerung eingeführt werden. Die hier beschriebene Architektur ist zunächst verteilt ausgelegt, wobei eine hybride oder vollständig zentrale ebenfalls vorstellbar ist und später in Abschnitt 4.4.1 vorgestellt wird. Der Emulationsgrad wird durch ein relativ einfaches, abstraktes Berechnungsmodell bestimmt.

Wie bereits in Abschnitt 2.2 erläutert, ist der Exklusivzugriff auf das gemeinsame Medium eines Netzes notwendig, um die Funktion des Netzes sicherzustellen. Es darf zu jedem Zeitpunkt höchstens eine Station auf dem Medium senden, da sonst Kollisionen auftreten, die das Signal zerstören und die Sendung somit beim Empfänger im allgemeinen nicht mehr korrekt interpretiert werden kann.

Durch den Exklusivzugriff teilen sich die am gemeinsamen Medium angeschlossenen Stationen die verfügbare physikalische Übertragungsrate. Die Summe der Übertragungsraten b_i der Stationen s_i zu einem bestimmten Zeitpunkt ist stets kleiner gleich der physikalischen Übertragungsrate b: $\sum_{i=1}^{N} b_i \leq b$, bei N Stationen. Diese Invariante erinnert an ähnliche geartete wie sie beispielsweise beim Kredit-Verfahren auftreten, einem verteilten Algorithmus, der sich zur Terminierungserkennung bzw. zur Garbage Collection in verteilten Systemen eignet.

Es bietet sich nun an, ein solches verteiltes Verfahren zu verwenden, um zunächst einmal die Übertragungsrate unter den Stationen aufzuteilen. Hierzu sei die physikalische Übertragungsrate b gegeben. Sie entspricht der Kreditsumme. Die einzelnen Kreditanteile, d.h. Teile b_i der Übertragungsrate, werden den einzelnen Stationen zugeschlagen. Eine Station darf nun maximal mit der ihr momentan zugeteilten Übertragungsrate senden. Da die Zuteilung der Anteile natürlich dynamisch vor sich gehen soll, ist ein Handel mit Kreditanteilen im Netz erforderlich. Analog zur Vorgehensweise bei Lastbalancierung läßt sich der Handel sender- oder empfängerinitiiert umsetzen. Im senderinitiierten Fall geben Stationen, die aktuell ihren Anteil nicht ausnutzen, einen Teil an den Markt ab. Bei empfängerinitiiertem Handel stellen Stationen, die mehr senden wollen als ihr Anteil zuläßt, eine Nachfrage auf dem Markt.

Das beschriebene Verfahren entspricht einem Medienzugriffsverfahren und zwar einem koordinierten, d.h. kollisionsfreien. Daraus folgt, daß vorerst nur diese eine bestimmte Klasse von MAC-Verfahren emulierbar ist. Es stellt sich die Frage, welche Klassen existieren und wie sich ihr Verhalten unterscheidet.

Die Familie der Verfahren zur Steuerung des Medienzugriffs läßt sich durch eine hierarchische Klassifizierung in drei grundlegende Klassen einteilen (Abb. 4.3). Der auffallendste Unterschied im Verhalten dieser Klassen liegt in der tatsächlich auf dem Netz verfügbaren Übertragungsrate und der Verzögerung bei der Gewährung des exklusiven Medienzugriffs [Sid99, RS90].

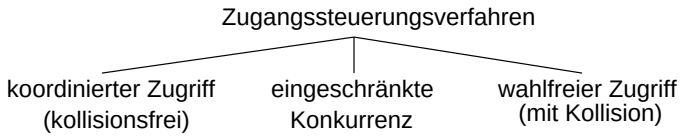


Abbildung 4.3: Oberste Hierarchiestufe der Klassifizierung von MAC-Verfahren

Bei kollisionsfreiem, koordiniertem Zugriff steigt mit wachsender Last auch die verfügbare Übertragungsrate bis maximal zur physikalischen Netzübertragungsrate. Allerdings kann je nach Verfahren, die Verzögerung beim Medienzugriff v.a. bei niedriger Last relativ hoch sein. Beispielsweise ist dies bei Token-Ring der Fall, wo eine Station im schlechtesten Fall beinahe eine ganze Ringumrundung warten muß, bis sie das Token erhält. Dies ist der Fall, wenn das Token zum Zeitpunkt eines Sendewunsches kurz zuvor bereits an den nächsten Ringnachbar weitergegeben wurde. Wenn eine einzige Token-Ring Station mit der vollen Übertragungsrate des Netzes senden möchte, so ist dies praktisch unmöglich, da sie nach Ablauf ihres Sendequantums das Token abgeben und warten muß, bis es einmal um den Ring gelaufen ist und wieder bei ihr ankommt.

Bei wahlfreiem Zugriff, der kollisionsbehaftet ist, geht bei höherer Last die verfügbare Übertragungsrate zurück, da durch Kollisionen Rahmen wiederholt gesendet werden müssen. Wiederholte Sendungen kosten einen Teil der physikalisch verfügbaren Übertragungsrate. Andererseits ist die Medienzugriffsverzögerung bei niedriger Last meist sehr gering. Im Falle von Verfahren ohne Trägererkennung (carrier sensing) wie z.B. ALOHA ist die initiale Verzögerung gleich Null, da in jedem Fall sofort gesendet wird. Die Verzögerung steigt jedoch mit wachsender Last durch häufige Kollisionen stark an.

Dazwischen gibt es noch die Klasse der Verfahren mit eingeschränkter Konkurrenz. Diese versuchen

durch einen Kompromiß sowohl schnellen Medienzugriff bei geringer Last als auch möglichst wenig Kollisionen bei hoher Last zu erreichen, indem sie die besten Eigenschaften aus den beiden vorhergehenden Klassen vereinigen. Hierzu wird die Wahrscheinlichkeit erneuter Sendeversuche nach Kollisionen innerhalb von Zeitschlitzen asymmetrisch über die Stationen verteilt [Tan96].

Um die verschiedenen Klassen für die Emulation abzudecken, läßt sich das oben beschriebene Kredit-Verfahren mit einem Modell, welches das Last-Verhalten eines bestimmten Netztyps beschreibt, überlagern. Statt des Kredit-Verfahrens lassen sich auch andere kollisionsfreie Verfahren einsetzen.

Der Eingangsparameter für das Last-Verhalten ist definiert durch die normalisierte Last G:

Definition 2 *In Anlehnung an [Sid99] entspricht die normalisierte Last G der Anzahl Sendeversuche pro Rahmenzeit T:* $G = g \cdot T = \Lambda \cdot T = \left(\sum_{i=1}^{N} \lambda_i\right) \cdot \frac{L}{C}$*, wobei die Parameter Last g, Rahmenzeit T in s/Rahmen, aggregierte Ankunftsrate* Λ *in Rahmen/s, Ankunftsrate* λ_i *einer Station i in Rahmen/s, Anzahl der Stationen N, Übertragungsrate C in Bit/s und Rahmengröße L in Bit/Rahmen verwendet werden.*

Die Netzlast G wird im emulierten Netz gemessen. Jede Station S_i kennt ihre Ankunftsrate λ_i. Um die aggregierte Ankunftsrate Λ des gesamten Netzes auf jedem Knoten bekanntzumachen, werden sämtliche Stationen über die Raten aller anderen Stationen unterrichtet, so daß $\Lambda = \sum_{i=1}^{N} \lambda_i$, bei N Stationen.

Aus der normalisierten Last G läßt sich der normalisierte Durchsatz S pro Rahmenzeit T sowie die normalisierte Verzögerung D mit Hilfe mathematischer Formeln oder vereinfacht durch Nachschlagen in vorberechneten oder experimentell ermittelten Tabellen erhalten. Berechnungsformeln sind beispielsweise in [Sid99, RS90, Tan96] zu finden.

Der sich ergebende Durchsatz ist in Abb. 4.4 für eine Auswahl verschiedener Medienzugriffsverfahren über der Last aufgetragen. Die verwendeten Parameter und Formeln befinden sich in Anhang A. Beim theoretischen, idealen Verfahren hängt der Durchsatz direkt linear von der Last ab. Der Definitionsbereich beschränkt sich sinnvollerweise allerdings auf maximal einen Senderversuch pro Rahmenzeit. Mehr Versuche wären sinnlos, da bereits der maximal mögliche Durchsatz erreicht ist. Bei den ALOHA-Verfahren ist deutlich zu erkennen wie bereits früh ein Sättigungspunkt erreicht wird und der Durchsatz anschließend gegen Null geht. 1-persistentes CSMA, wie es bei Ethernet – dort allerdings mit Kollisionserkennung (/CD) – zum Einsatz kommt, erreicht einen besseren Durchsatz als ALOHA, was auf die Trägererkennung zurückzuführen ist. Hierdurch können unnötige Kollisionen durch Senden, während ein anderer Rahmen sich bereits sichtbar in der Übertragung befindet, vermieden werden. Nonpersistente CSMA-Verfahren weisen einen langsamen Anstieg des Durchsatzes auf und erreichen ihren Sättigungspunkt bei höherer Last.

Zur Anwendung auf das vorgeschlagene Überlagerungsverfahren wird die Sendeübertragungsrate des Netzes auf den bei der aktuell herrschenden Last verfügbaren Durchsatz limitiert. Dies bedeutet, die Kreditsumme nimmt entsprechend ab. Die limitierte Übertragungsrate b_{lim} ist das Produkt aus normalisiertem Durchsatz S und der durch Kredite zugeschlagenen Teilübertragungsrate b_{kred}: $b_{\text{lim}} = S \cdot b_{\text{kred}}$.

Der zweite wichtige Kennwert der verschiedenen Medienzugriffsverfahren ist die Verzögerung vom Sendewunsch eines Rahmens bis zum Start seiner erfolgreichen Übertragung. Die normalisierte Verzögerung läßt sich analog zum Durchsatz wie oben beschrieben ermitteln.

Die Anwendung auf das Überlagerungsverfahren findet durch eine zusätzliche Verzögerung einer jeden Rahmensendung auf den Knoten statt. Die momentan gültige Zusatzverzögerung t_d ergibt sich aus dem Produkt der aktuellen normalisierten Verzögerung D und der Rahmenzeit T: $t_d = D \cdot T$.

Im Sinne des allgemeinen Emulationsmodells aus Definition 1 wurde also ein Eingabeparameter identifiziert:

e_1 momentane Netzlast G.

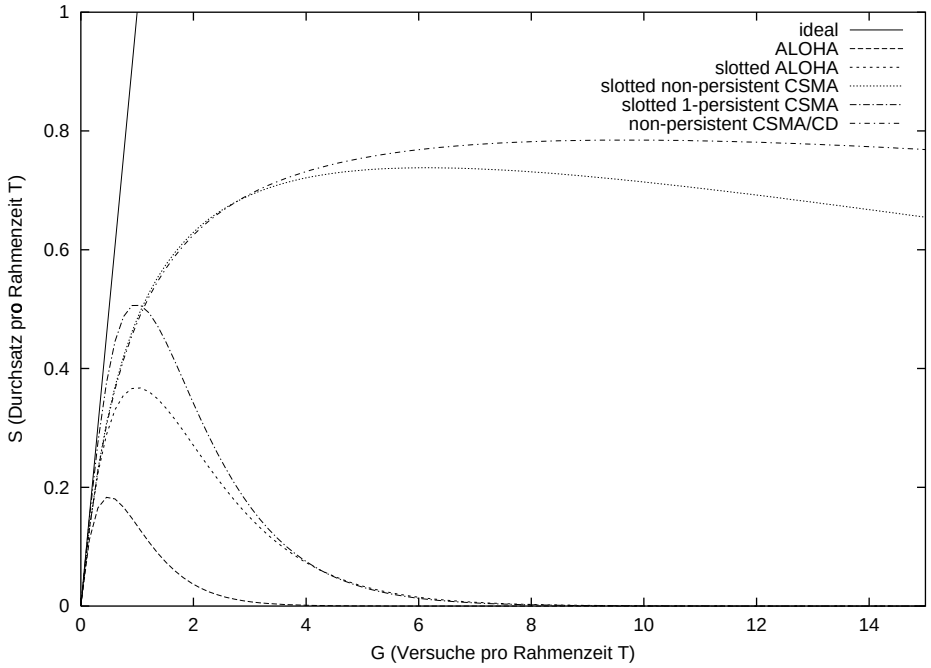


Abbildung 4.4: Kennlinien für Durchsatz in Abhängigkeit von der Netzlast

Die Ausgabeparameter sind:

a_1 momentan verfügbarer Durchsatz S,

a_2 momentane durchschnittliche Verzögerung beim Medienzugriff D.

Die Abbildung ist definiert durch:

$$\begin{aligned} a_1 &:= S(G), \\ a_2 &:= D(G). \end{aligned}$$

4.3 Geringer Emulationsgrad

Als Beispiele für Emulationskonzepte mit geringem Emulationsgrad folgen zwei Konzepte, welche die gängigen LAN-Typen Ethernet und Token-Ring durch Nachbildung der MAC-Schicht in Software emulieren. Die Implementierung erfolgt im Systemmodus auf Sicherungsschicht. Aufgrund der Nachbildung des Medienzugriffsverfahrens weist das Konzept dieselbe Architektur wie die der nachgebildeten Verfahren selbst auf, d.h. es handelt sich um eine verteilte Architektur.

4.3.1 Ethernet

Die gegebene NET-Emulationsumgebung verwendet einen Ethernet-Switch zur Konfiguration von Topologien mit Hilfe von VLANs. Über diesen Switch geht der gesamte Emulationsverkehr, der von den Testanwendungen erzeugt und verbraucht wird. Das Primärmerkmal eines Switch ist die Verhinderung sämtlicher Kollisionen durch Direktverbindung von Knotenpaaren und eventuelles Zwischenpuffern von

Rahmen. Insofern überführt der Switch also ein Netz mit gemeinsamem Medium in eine Menge von Punkt-zu-Punkt Verbindungen zwischen dem Switch und den verschiedenen Stationen. Es entsteht eine Topologie mit Sternverkabelung.

Nun soll aber ein Netz mit gemeinsamem Medium emuliert werden, also kann man versuchen die oben beschriebene, nun unerwünschte Switch-Eigenschaft „rückgängig“ zu machen. Da sich im derzeitig eingesetzten Switch nur ein Port als Monitor, der alle Rahmen aus einem bestimmten VLAN zugestellt bekommt, konfigurieren läßt und um außerdem von den Optionen verschiedener Switches unabhängig zu sein, muß die Emulations-Software selbst dafür Sorge tragen, daß grundsätzlich jeder gesendete Rahmen von allen Stationen innerhalb eines VLANs empfangen werden kann. Zu diesem Zweck wird jeder zu sendende Rahmen in einem Ethernet-Broadcast-Rahmen getunnelt. Abb. 4.5 zeigt wie der eigentlich bereits zum Senden bereite Rahmen mit der spezifischen Quell- und Zieladresse als Nutzlast in einen neuen Rahmen eingepackt wird. Dieser neue Rahmen hat als Ziel die Ethernet-Broadcast MAC-Adresse FF:FF:FF:FF:FF:FF. Auf diese Weise wird der Switch gezwungen, den Rahmen an jeden Port innerhalb desselben VLANs zuzustellen. Andererseits wird durch die Tunnelung der notwendige Original-Rahmenkopf erhalten.

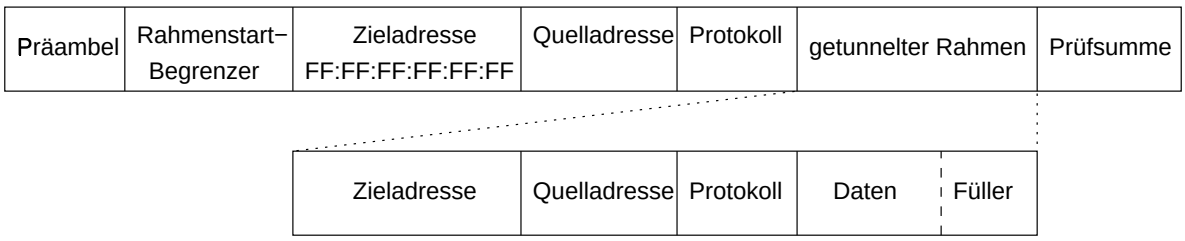


Abbildung 4.5: Ethernet-Rahmenformat für die Tunnelung von Rahmen in Broadcast-Sendungen

Zwar ist nun eine Art gemeinsames Medium wiederhergestellt worden, jedoch verhindert die Pufferung im Switch nach wie vor Kollisionen. Um hierfür eine Lösung zu finden, wird die bereits in Kapitel 4 angesprochene Annahme zugrundegelegt, daß die Übermittlungszeiten für die Broadcast-Rahmen vernachlässigbar klein gegen die durch die Emulation eingeführten Verzögerungen sind. Somit kann jede Station beim Empfang eines Rahmens annehmen, daß der Rahmen nach ihrer lokalen Zeit genau dann gesendet wurde, als sie ihn selbst empfangen hat. Auf diese Weise können Stationen Zeitstempel für empfangene Rahmen anhand ihrer lokalen Uhr vergeben, ohne die Uhren der Stationen untereinander synchronisieren zu müssen.

Behauptung 1 *Unter der Annahme, daß Übertragungszeiten in der Emulationsumgebung vernachlässigbar sind, kann jede Station unabhängig, allein mit Hilfe ihrer lokalen Zeit für empfangene und zu sendende Rahmen sowohl Träger- als auch Kollisionserkennung durchführen sowie Übertragungsratenlimitierung und Übertragungsverzögerung nachbilden.*

Hierzu sind die zwei folgenden Zeitwerte von Belang:

Signalausbreitungszeit t_s: Sie ist abhängig von der Signalausbreitungsgeschwindigkeit im Medium $v_c \approx \frac{2}{3}v_0$ und der Netztopologie, im Speziellen vom Abstand d zwischen Quell- und Zielstation.

$$t_s = \frac{d}{v_c} = \frac{3d}{2v_0} = \frac{d}{2 \cdot 10^8 m/s} \tag{4.1}$$

Rahmensendezeit t_r: Sie ist abhängig von der Rahmengröße r und der Übertragungsrate b.

$$t_r = \frac{r}{b} \tag{4.2}$$

Jede Station besitzt die folgenden lokalen Variablen:

t aktuelle lokale Zeit der Station ($t > 0$)

ts Zeitpunkt, zu dem von der Station ein Träger auf dem Medium und somit der Beginn eines Rahmens erkannt wird.

tsr Zeitpunkt, zu dem der Träger auf dem Medium verschwunden ist und somit die Station das Ende eines Rahmens erkannt hat.

kollision Boolescher Wert, der anzeigt, ob in der Zukunft eine Kollision passieren wird.

sendend Boolescher Wert, der anzeigt, ob die Station selbst momentan einen Rahmen sendet.

aktiv Verweis auf den momentan an dieser Station „aktiven“ Rahmen. Es handelt sich dabei um einen Rahmen, der entweder für die spätere Auslieferung an die Vermittlungsschicht vorgesehen ist oder um einen Rahmen, der von dieser Station momentan gesendet wird (*sendend* = *wahr*). Falls kein aktiver Rahmen vorliegt, ist der Verweis Null.

t_s[1...N] Eindimensionales Feld mit den Signalausbreitungszeiten zwischen dieser Station und der jeweils durch den Index angegebenen anderen Station. Die Zeiten ergeben sich mit Hilfe von Gleichung 4.1 aus der Topologiebeschreibung des zu emulierenden Netzes.

Außerdem wird eine Pseudostruktur für Rahmen verwendet. Sie besitzt die folgenden Selektoren, die zu ihrer Referenzierung, jeweils durch einen Punkt getrennt, an die Strukturvariable angehängt werden:

t Zeitstempel, zu dem der Rahmen empfangen wurde.

sender Absendestation des Rahmens.

länge Rahmenlänge inklusive Rahmenkopf und Nutzlast.

kollisionszahl Anzahl der Kollisionen, in die ein zu sendender Rahmen bereits verwickelt wurde. Sie bestimmt das Contentionslot-Intervall beim binären Backoff-Verfahren zur Bestimmung erneuter Sendeversuche. Initial ist *kollisionszahl* = *0*.

Jede Station versieht empfangene Rahmen unmittelbar mit einem Zeitstempel *rahmen.t* nach ihrer lokalen Uhr *t*.

Die Beispiele in den folgenden Abschnitten basieren auf einer Netztopologie wie in Abb. 4.6 dargestellt. Die Signalausbreitungszeit zwischen Station S1 und S2 beträgt zwei Zeiteinheiten, zwischen S2 und S3 eine Zeiteinheit. Wird der Bus als maximal lang angenommen, so ergibt sich eine minimale Rahmenlänge deren Sendezeit $\tau = 6$ Zeiteinheiten beträgt. Dies entspricht der doppelten Signalausbreitungszeit von einem Busende zum anderen. Daraus ergibt sich für die Fallbeispiele ein Contentionslot von 6 Zeiteinheiten Dauer. Der Einfachheit halber wurde die Zeitdauer τ_{jam}, während der Jambits nach einer erkannten Kollision gesendet werden, ebenfalls zu $\tau_{jam} = 6$ angenommen. Dies ist zulässig, da es sich um die dafür minimale Dauer handelt.

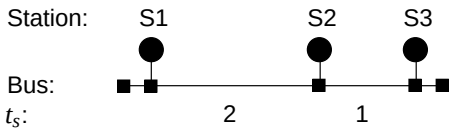


Abbildung 4.6: Ethernet-Netztopologie für Fallbeispiele

Trägererkennung (carrier sensing)

Möchte eine Station einen Rahmen senden und ist $ts \leq t < tsr$, dann befindet sich noch ein anderer Rahmen in der Übertragung. Das heißt ein Träger wurde erkannt und es darf nicht gesendet werden. Ein erneuter Sendeversuch wird aufgrund der 1-Persistenz von Ethernet für den Zeitpunkt *tsr* vorgemerkt.

In allen anderen Fällen darf die Station sofort senden.

Abb. 4.7 zeigt verschiedene Fälle mit bzw. ohne erkannten Träger in einem Raum-Zeit-Diagramm. Die Tatsache, daß die lokalen Uhren der Stationen untereinander nicht synchronisiert sind, ist durch unterschiedliche Startzeitpunkte für *t* angedeutet. Es wird eine streng monoton steigende diskrete Zeit verwendet, die ihren Ausdruck in natürlichen Zahlen als Zeitwerte findet. Weiterhin findet die Annahme Verwendung, daß die Nachrichtenübermittlungszeit in der Emulationsumgebung vernachlässigbar ist. Dadurch stehen die Pfeile für transportierte Rahmen senkrecht zur Zeitachse und haben keine zeitliche Ausdehnung. Der Träger ist jeweils durch einen schraffierten, rechteckigen Bereich auf der Zeitachse gekennzeichnet. Ereignisse, wie beispielsweise ein Sendevorgang, sind durch ausgefüllte Kreise auf der Zeitachse dargestellt und werden durch einen darüberstehenden Text beschrieben. Die Variablenwerte für wahr, falsch und Null sind durch T, F und - im Diagramm abgekürzt.

Alle Stationen befinden sich zu Beginn in einem initialen Zustand, zu dem die lokalen Variablen wie folgt vorbelegt sind: $ts = tsr = 0$, $kollision = sendend = falsch$, $aktiv = Null$. Station S1 sendet zum Zeitpunkt $t = 4$ einen Rahmen m1. S1 aktualisiert ihren Zustand in den lokalen Variablen: Der Rahmen m1 wird von $ts := t = 4$ bis $tsr := ts + t_r = 10$ von ihr selbst ($sendend = wahr$) gesendet und wird aus ihrer Sicht momentan auf dem Netzmedium übertragen ($aktiv := m1$). Station S2 empfängt den Rahmen m1 bei $t = 2$, setzt dessen Zeitstempel $m1.t := 2$ und berechnet ihre lokalen Variablen neu: $ts := m1.t + t\_s[m1.sender] = 4$, $tsr := ts + t\_r(m1.länge) = 10$, $aktiv := m1$. Die Funktion $t\_r()$ berechnet die Rahmensendedauer für die als Argument übergebene Rahmenlänge entsprechend Gleichung 4.2. Station S3 verfährt entsprechend, wobei die unterschiedliche Signalausbreitungsdauer t_s von S1 zu S2 bzw. zu S3 durch verschiedene Abstände zwischen Rahmenempfang und Trägerbeginn zum Ausdruck kommt.

Bei Station S1 ist zum Zeitpunkt $t = 6$ ein potentieller Sendewunsch durch ein neues Paket aus der Vermittlungsschicht abgebildet. Da sie selbst momentan einen Rahmen sendet, muß der Sendewunsch zunächst verzögert werden. Um diesen Spezialfall, daß selbst gesendet wird, nicht gesondert behandeln zu müssen, läßt sich der momentane Träger allgemein dadurch erkennen, daß der Zeitpunkt $t = 6$ zwischen *ts* und *tsr* liegt. Analog erkennt Station S3 zum Zeitpunkt $t = 12$ einen Träger auf dem Medium und muß ihren potentiellen Sendewunsch bis $tsr = 13$ verzögern. Station S2 könnte zum Zeitpunkt $t = 3$ einen Sendewunsch erfüllen, da hier noch kein Träger von m1 erkannt wird. Eine Sendung zu diesem Zeitpunkt würde später zu einer Kollision führen.

Kollisionserkennung (collision detection)

Empfängt eine Station einen Rahmen mit $rahmen.t + t\_s[rahmen.sender] < tsr$, dann überschneidet sich der Trägerbeginn des empfangenen Rahmens mit einem Träger eines anderen aktiven Rahmens. Das

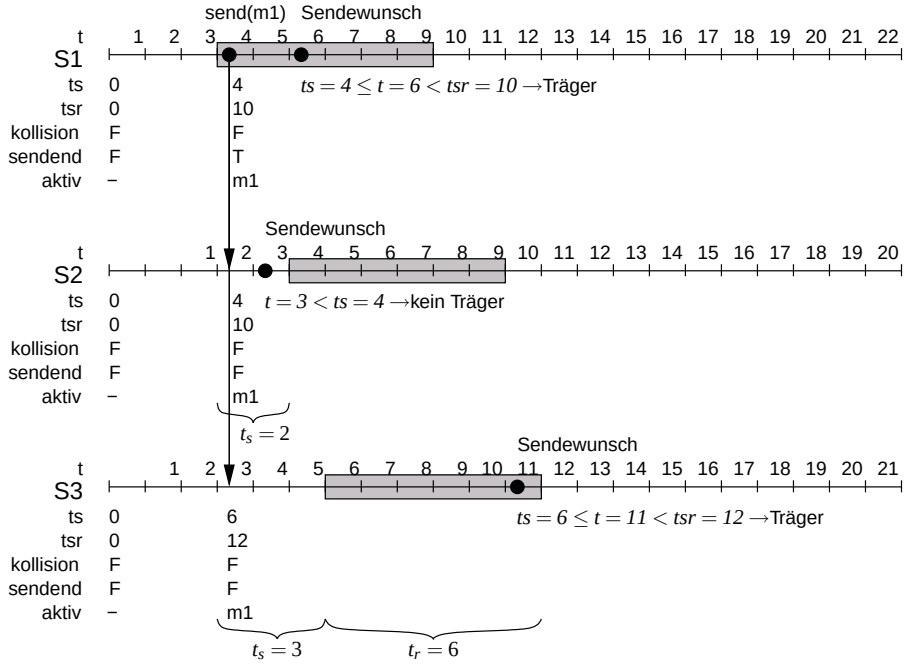


Abbildung 4.7: Trägererkennung

heißt es wird eine Kollision geben, die bei *kollisionsbeginn* := $max\{rahmen.t + t\_s[rahmen.sender], ts\}$ beginnt und bei *kollisionsende* := *kollisionsbeginn* + τ_{jam} enden wird. Die lokalen Variablen werden aktualisiert: *kollision* := *wahr*, *ts* := $min\{rahmen.t + t\_s[rahmen.sender], ts\}$.

Falls es sich um die erste Verursachung einer Kollision handelt, d.h. *kollision* war zuvor falsch, wird *tsr* := *kollisionsende* gesetzt, da zu diesem Zeitpunkt der bzw. die Träger vom Medium verschwunden sein werden und das Medium somit wieder frei sein wird. Falls es sich nicht um die erste Kollisionsverursachung handelt, wird *tsr* := $max\{tsr, kollisionsende\}$ gesetzt, um auch die späteste Verursachung, die die Kollisionsdauer verlängert, zu berücksichtigen. In jedem Fall wird *rahmen.kollisionszahl* inkrementiert.

War nun die Station selbst aktiv an der Kollision beteiligt, d.h. *sendend* = *wahr*, dann wird *sendend* := *falsch* und *aktiv* := *Null* gesetzt, d.h. das Senden sofort abgebrochen und ein erneuter Sendeversuch gemäß dem binären Backoff-Verfahren beginnend ab *tsr* eingeplant. Bei Überschreitung der Versuchsanzahl wird stattdessen mit einem Fehler abgebrochen. War die Station nur passiv beteiligt (*sendend* = *falsch*), dann wird der empfangene sowie der aktive Rahmen einfach verworfen und *aktiv* := *Null* gesetzt.

Möchte eine Station einen Rahmen bei $t < ts$ senden, dann ist bereits klar, daß eine Kollision ab *ts* passieren wird. Allerdings ist der andere Rahmen, der hier ankommen wird, noch unterwegs und die Station darf diese Kenntnis der Zukunft noch nicht ausnutzen. Deshalb wird durch *tsr* := $max\{ts + \tau_{jam}, tsr\}$ das späteste Kollisionsende berechnet, anschließend der eigene Rahmen gesendet, *kollision* := *wahr* gesetzt und *ts* := *t* aktualisiert. Außerdem wird ein evtl. aktiver Rahmen verworfen und *aktiv* := *Null* gesetzt. Ein erneuter Sendeversuch wird gemäß dem binären Backoff-Verfahren beginnend ab *tsr* eingeplant. Bei Überschreitung der Versuchsanzahl wird stattdessen mit einem Fehler abgebrochen.

In Abb. 4.8 ist ein Szenario dargestellt, in dem drei Stationen jeweils einen Rahmen senden, während noch kein Träger auf dem Medium zu erkennen ist. Dies führt hier im Beispiel zur Kollision der drei

Rahmen. Station S1 sendet einen Rahmen m1 zum Zeitpunkt $t = 4$. Alle Stationen aktualisieren ihre lokalen Variablen wie bereits in der Beschreibung zu Abb. 4.7 erläutert.

Station S2 nimmt bei $t = 2$ ihre Chance, einen Sendewunsch zu erfüllen, wahr und sendet Rahmen m2 aus. Da $t < ts$, ist bereits klar, daß dies zu einer Kollision führen wird ($kollision := wahr$). S2 stellt voraussehend das „Senden“ ein ($sendend := falsch$), berechnet das Kollisionsende $tsr := ts + \tau_{jam} = 10$, verwirft den bisher aktiven Rahmen m1 ($aktiv := Null$) und setzt zuletzt $ts := 3$ vor. Ein erneuter Sendeversuch von m2 wird für den ersten Contentionslot eingeplant, d.h. bei $t = tsr = 10$.

Die Station S1 empfängt m2 bei $t = 5$ und stempelt $m2.t := 5$. Wegen $m2.t + t\_s[m2.sender] = 7 < tsr = 10$ wird eine Kollision erkannt, an der S1 aktiv beteiligt ist ($sendend = wahr$) und deshalb sofort das „Senden“ abbricht ($sendend := falsch$) und $aktiv := Null$ setzt. Anhand der Hilfsvariablen $kollisionsbeginn := m2.t + t\_s[m2.sender] = 7$ und $kollisionsende := 13$ wird $tsr := kollisionsende = 13$ gesetzt. Ein erneuter Sendeversuch für den Rahmen m1 wird zunächst für den ersten oder zweiten Contentionslot ($m1.kollisionszahl = 1$) ab tsr eingeplant.

Station S3 empfängt m2 bei $t = 4$ und stempelt $m2.t := 4$. Wegen $m2.t + t\_s[m2.sender] = 5 < tsr = 12$ wird eine Kollision erkannt, an der S3 passiv beteiligt ist ($sendend = falsch$). Deshalb wird m2 und der aktive Rahmen verworfen und $aktiv := Null$ gesetzt. Die Kollision wird bei $t = ts = 6$ beginnen und bei $tsr := 12$ beendet sein. Weitere lokale Variablen werden aktualisiert: $kollision := wahr$, $ts := m2.t + t\_s[m2.sender] = 5$. Noch im selben Zeitschritt $t = 4$ gibt S3 einem Sendewunsch nach und sendet m3 aus. Auch hier ist wiederum wie bei m2 auf S2 wegen $t = 4 < ts = 5$ klar, daß eine zusätzliche Kollision folgen wird. Diese Kollision würde bei $t = ts = 5$ beginnen und bei $t = 11$ beendet sein. Da dieses Kollisionsende nicht das späteste ist, bleibt $tsr = 12$ erhalten. Zuletzt wird $ts := t = 4$ vorgesetzt und ein erneuter Sendeversuch von m3 für den zweiten Contentionslot bei $t = tsr + (2-1) \cdot \tau = 18$ eingeplant.

Bei S2 führt der Empfang von m3 bei $t = 3$ zu keinen weiteren Zustandsveränderungen, da m3 zur gleichen Zeit eine Kollision verursacht wie der bereits verarbeitete Rahmen m1. S1 dagegen, empfängt m3 bei $t = 5$ und stempelt $m3.t := 5$. Wegen $m3.t + t\_s[m3.sender] = 8 < tsr = 13$ wird eine Kollision erkannt, an der S1 nach wie vor aktiv beteiligt ist[1]. Anhand der Hilfsvariablen $kollisionsbeginn := m3.t + t\_s[m3.sender] = 8$ und $kollisionsende := 14$ ergibt sich nun aber ein neuer Zeitpunkt für das späteste Kollisionsende: $tsr := 14$. Dies bedeutet, der erneute Sendeversuch für m1 wird für den zweiten Contentionslot bei $t = tsr + (2-1) \cdot \tau = 20$ neu gewählt.

Station S2 hat ihren erneuten Sendeversuch von m2 für den ersten Contentionslot eingeplant, wohingegen die beiden anderen Stationen S1 und S3 den zweiten Slot vorgesehen haben. Da sich S2 allein um diesen ersten Zeitschlitz bewirbt, wird der erneute Sendeversuch erfolgreich ohne Kollision verlaufen. Die Sendewiederholungen von S1 und S3 können dagegen später eine erneute Kollision verursachen.

Erfolgreicher Rahmenempfang

Ein erfolgreicher Rahmenempfang beginnt damit, daß sich der Träger nicht mit dem eines anderen Rahmens zeitlich überlappt, d.h. $rahmen.t + t\_s[rahmen.sender] \geq tsr$. Also ist zunächst keine Kollision abzusehen. Die lokalen Variablen der Station werden wie folgt neu berechnet: $ts := rahmen.t + t\_s[rahmen.sender]$, $tsr := ts + t\_r(rahmen.länge)$, $aktiv := rahmen$. Die Funktion $t\_r()$ berechnet für die übergebene Rahmenlänge die Rahmensendezeit t_r nach Gleichung 4.2. Die verzögerte Auslieferung an die Vermittlungsschicht wird für tsr eingeplant. Wird der empfangene und nun aktive Rahmen in keine Kollision verwickelt, so wird er schließlich ausgeliefert und der Empfang war erfolgreich. Abschließend wird $aktiv := Null$ gesetzt.

[1] Auch wenn nun bereits von der vorherigen Kollision $sendend = falsch$ ist, so existiert doch ein eingeplanter erneuter Sendeversuch für m1.

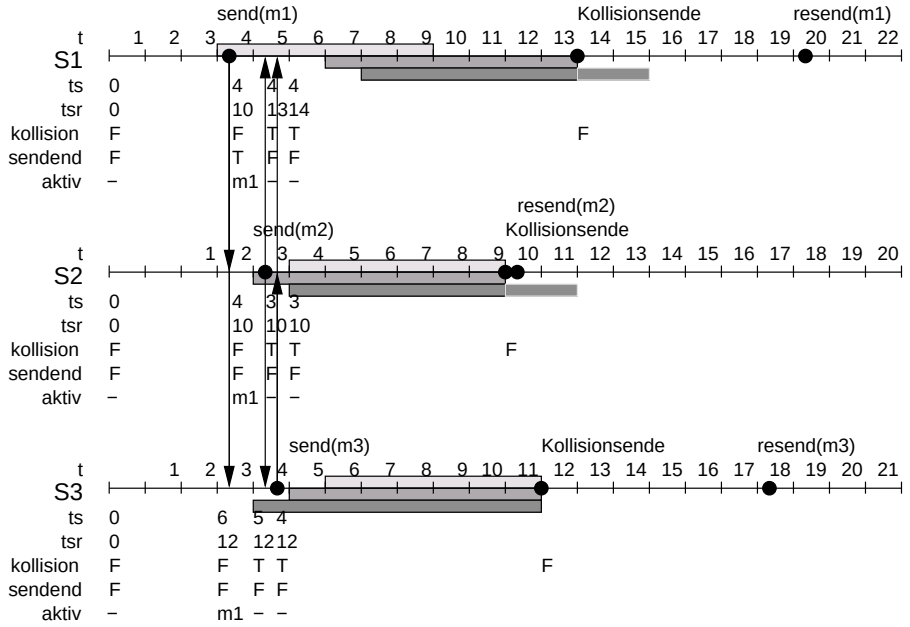


Abbildung 4.8: Kollisionserkennung

Abb. 4.9 veranschaulicht einen erfolgreichen Rahmenempfang. Station S1 sendet bei $t = 4$ einen Rahmen m1 aus. Die lokalen Variablen auf den Stationen berechnen sich wie bereits in der Beschreibung zu Abb. 4.7 erläutert. S2 liefert den empfangenen Rahmen m1 bei $t = tsr = 10$ an ihre Vermittlungsschicht aus. Station S3 liefert m1 bei $t = tsr = 12$ an ihre Vermittlungsschicht aus.

Erfolgreiche Rahmensendung

Eine erfolgreiche Rahmensendung beginnt damit, daß kein Träger auf dem Medium erkannt wird – auch nicht in absehbarer Zukunft ($t \not< ts$) –, d.h. $t \geq tsr$. Die lokalen Variablen werden wie folgt neu berechnet: *sendend* := *wahr*, $ts := t$, $tsr := ts + t\_r(rahmen.länge)$, *aktiv* := *rahmen*. Anschließend wird der Rahmen ausgesandt. Wird der in Sendung befindliche und nun aktive Rahmen in keine Kollision verwickelt, so war die Sendung erfolgreich. Abschließend wird zum Zeitpunkt *tsr* *sendend* := *falsch* und *aktiv* := *Null* gesetzt.

Zwei erfolgreiche Rahmensendungen sind in Abb. 4.10 dargestellt. S1 sendet den Rahmen m1 wie bereits bei Abb. 4.7 beschrieben. Während Station S2 den Rahmen m1 „empfängt“, erhält sie einen Sendewunsch aus ihrer Vermittlungsschicht bei $t = 5$. Da $ts = 4 \leq (t = 5) < tsr = 10$ ist, wird ein Träger erkannt und der Sendevorgang zunächst bis *tsr* verschoben. Bei $t = tsr = 10$ sendet S2 direkt im Anschluß an den empfangenen Rahmen m1 ihren eigenen Rahmen m2. Die Ankunft von m2 bei Station S1 ist problemlos, da der letzte Träger von m1 bereits bei $tsr = 10$ verschwunden war und der Sendeerfolg dort eintrat (*sendend* := *falsch*, *aktiv* := *Null*). Die Station S3 dagegen erhält m2 bei $t = 11$ und stempelt $m2.t := 11$. Zu diesem Zeitpunkt wird aber noch der Rest von Rahmen m1 „empfangen“ ($tsr = 12$). Diese Tatsache führt jedoch hier nicht zu einer Kollision, da zu $m2.t$ noch die Signalausbreitungszeit $t_s = 1$ zu addieren ist, so daß der Träger von m2 tatsächlich erst bei $m2.t + t\_s[m2.sender] = 12 \not< tsr$ bei S3 beginnt. Dies bedeutet, daß bei Station S3 der Rahmen m2 direkt im Anschluß an m1 erscheint. Schließlich stellt Station S2 ihren Sendeerfolg für Rahmen m2 bei $tsr = 17$ fest (*sendend* := *falsch*, *aktiv* := *Null*).

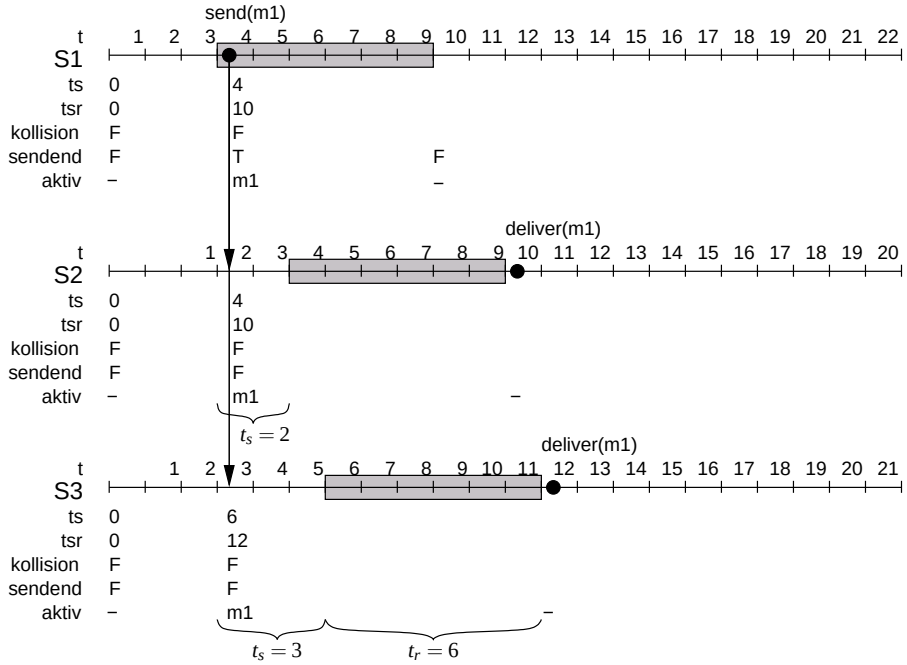


Abbildung 4.9: Erfolgreicher Rahmenempfang

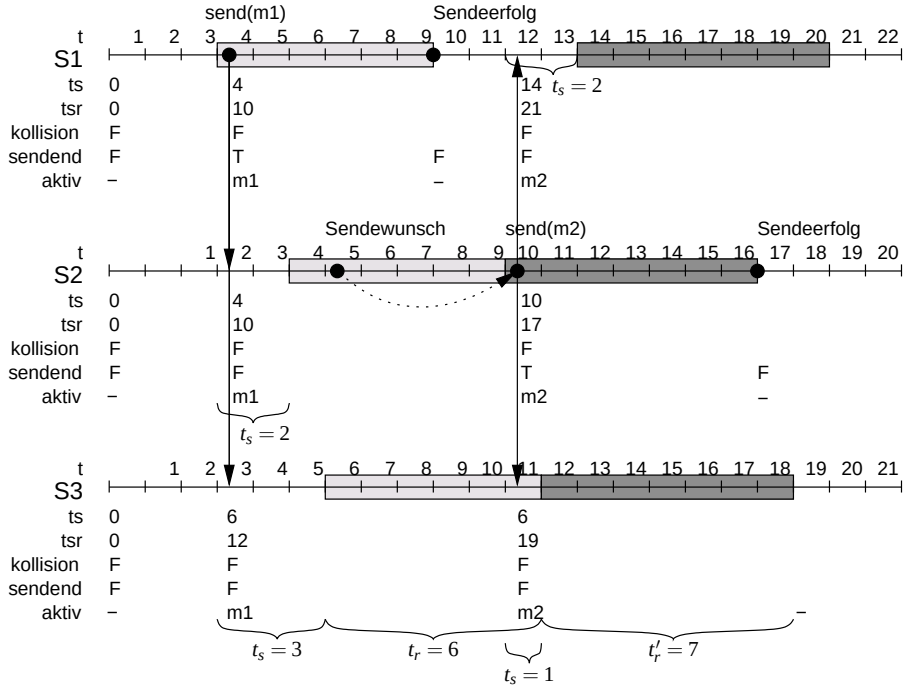


Abbildung 4.10: Erfolgreiche Rahmensendung

Hiermit wurde gezeigt, daß Träger- und Kollisionserkennung sowie Übertragungsratenlimitierung und Übertragungsverzögerung emulierbar sind. □

4.3.2 Token-Ring

Für die Emulation von Token-Ring bietet es sich an, statt eines allgemeineren Modells das Token-Verfahren zur Medienzugriffssteuerung direkt als Algorithmus in die Emulation zu übernehmen. Die damit nachbildbaren Verzögerungen beim Medienzugriff sind außerdem genauer als stochastische Ansätze [HLR02]. Bleibt also noch für die Limitierung der Übertragungsrate und auf der Topologie basierenden Verzögerungen zu sorgen.

Betrachtet man die Ringtopologie und läßt für einen Augenblick das Token-Verfahren außer Acht, so läßt sich das Netz als Menge von Punkt-zu-Punkt Verbindungen zwischen benachbarten Stationen darstellen. Zunächst könnte man annehmen, diese Verbindungen ließen sich durch Übertragungsrate und Verzögerung hinreichend beschreiben. Dann ließen sich die vorhandenen NETShaper zum Nachbilden der Nachbarverbindungen wiederverwenden. Bei näherer Betrachtung zeigt sich jedoch, daß die Stationen bei Token-Ring die nicht an sie adressierten Daten mit mindestens einem Bit Verzögerung bitweise und eben nicht rahmenweise auf ihrem Ausgang weitersenden. Abb. 4.11 zeigt schematisch die Funktionsweise einer Station im Kopiermodus.

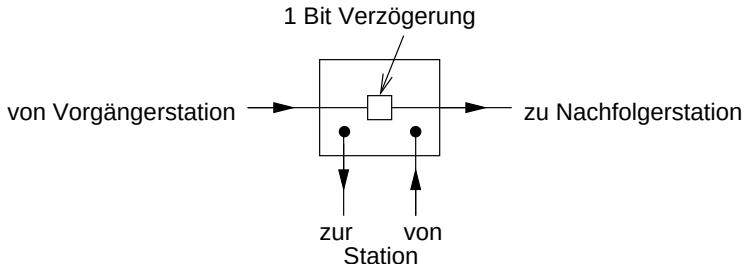


Abbildung 4.11: Modell einer Token-Ring Station im Kopiermodus

Wann ein Empfänger einen Rahmen in der Emulation empfängt, hängt von der Topologie ab – im Speziellen vom Abstand zwischen Sender und Empfänger. Bei Multi- und Broadcast-Sendungen erhalten die Empfänger der adressierten Gruppe die Nachricht zu jeweils unterschiedlichen Zeiten. Für den allgemeinen Fall bietet es sich also an, die Auslieferung von Rahmen beim Empfänger entsprechend zu verzögern. Hierzu werden die bereits in Abschnitt 4.3.1 eingeführten Zeitwerte Signalausbreitungs- t_s und Rahmensendezeit t_r (Gleichungen 4.1 und 4.2 auf Seite 23) herangezogen. Allerdings fließen nun in t_s neben dem reinen Stationsabstand auch die Bit-Verzögerungen in den zwischen Sender und Empfänger liegenden Stationen mit ein. Die notwendigen Informationen hierzu ergeben sich entweder aus der initialen Topologiebeschreibung des Netzes oder im Falle dynamischer Monitorstations-Wahl aus der aktuellen Topologie.

Ein Beispielszenario zeigt Abb. 4.12. Die darin verwendeten Größen werden in Tab. 4.1 erläutert. Als Netztyp liegt ein Token-Ring mit einer Übertragungsrate von $b = 4$ MBit/s zugrunde. Station S3 ist Monitor und fügt zusätzlich 4 Bit Verzögerung ein, um eine Ringbitzahl von 24 zu erreichen, so daß das Token von 3 Byte Größe auf dem Ring Platz findet und zirkulieren kann. Station S1 ist im Besitz des Tokens und sendet einen Broadcast-Rahmen der Größe 1500 Bit zum Zeitpunkt t. Die Sendung erfolgt als Broadcast im Netz der Emulationsumgebung, wobei die dafür notwendige Übertragungszeit vernachlässigt wird. Die jeweiligen Verzögerungen relativ zu t, nach denen die anderen Stationen den Rahmen ausliefern, ergeben sich aus der Signalausbreitungszeit $t_{s,j}[1]$ vom Sender S1 zum jeweiligen

Empfänger Sj und der Rahmensendezeit t_r, nach der das letzte Bit und somit der komplette Rahmen gemäß der Emulation empfangen wurde.

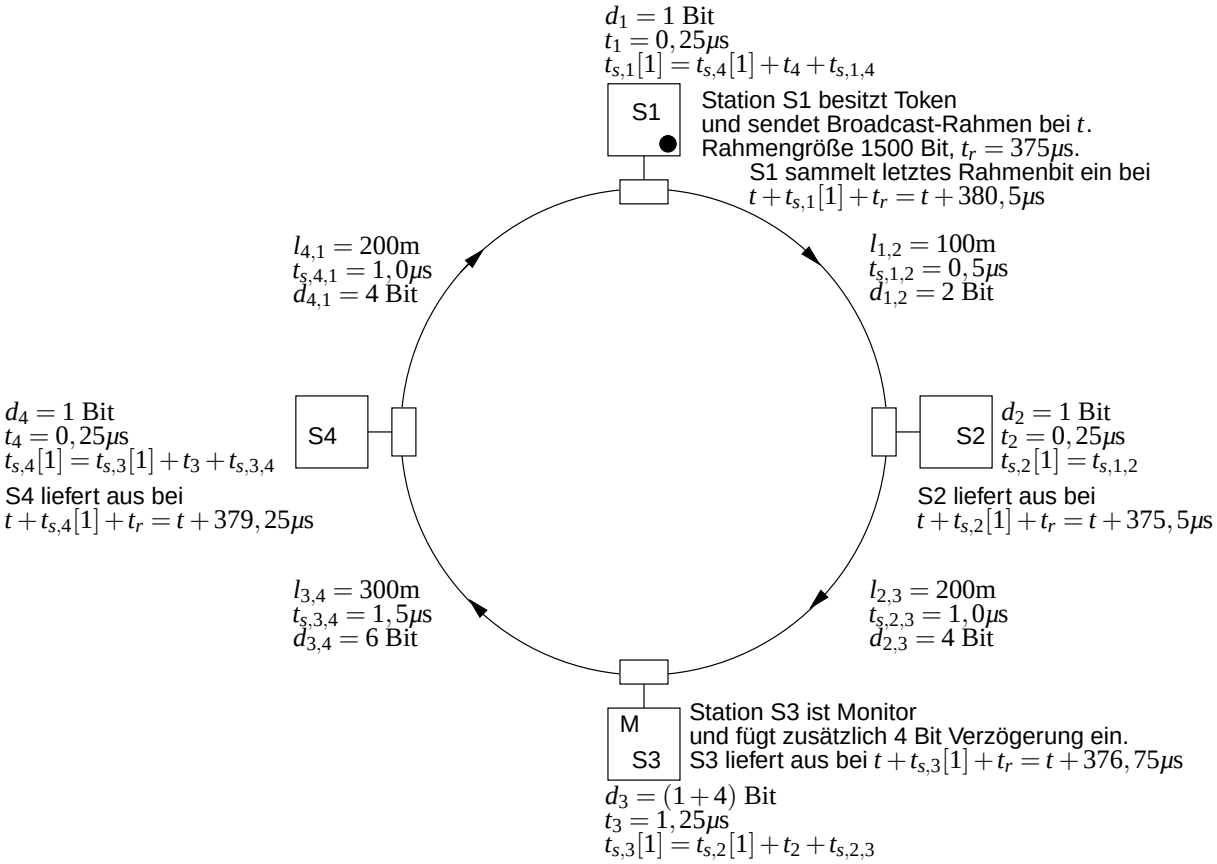


Abbildung 4.12: Unterschiedliche Auslieferungszeiten bei Multi- bzw. Broadcast

Die Signalausbreitungszeiten $t_{s,j}[i]$ brauchen für eine bestimmte Topologie auf jeder Station nur einmal gemäß der folgenden Gleichung 4.3 berechnet zu werden und können dann bei Bedarf sehr schnell aus einer Tabelle abgelesen werden.

$$t_{s,j}[i] = \sum_{m=i}^{j\ominus 1} t_{s,m,m\oplus 1} + \sum_{m=i\oplus 1}^{j\ominus 1} t_m \tag{4.3}$$

Die Zählvariable m wird modulo der Stationsanzahl N plus eins gezählt.

Auf analoge Art und Weise können weitere auf Token-Passing basierende Medienzugriffsverfahren wie beispielsweise Token-Bus oder FDDI emuliert werden. Darüber hinaus ist prinzipiell durch Nachbildung der MAC-Schicht in Software die Emulation eines jeden Medienzugriffsverfahren möglich. Dies erfordert dann aber natürlich andere, verschiedene Algorithmen und stellt neue Anforderungen an die Emulationsumgebung, z.B. in Bezug auf Echtzeitfähigkeit der Emulations-Software.

4.4 Zentraler Emulationsknoten

Der große Vorteil einer zentralen Emulationslösung ist der darin verwendbare Begriff einer globalen Zeit in einem ansonsten asynchronen, verteilen System. Ein zentraler Emulationsknoten ermöglicht die

Größe	Beschreibung	Einheit
	Eigenschaften von Station S*i*	
d_i	Bit-Kopierverzögerung in Station S*i*	Bit
t_i	Bit-Kopierverzögerung in Station S*i*	s
$t_{s,j}[i]$	Signallaufzeit von Station S*i* zu S*j*	s
	Netztopologie	
$l_{i,j}$	Kabellänge des Ringabschnitts zwischen S*i* und S*j*	m
$t_{s,i,j}$	Signallaufzeit vom einen Ringabschnittsende zum anderen	s
$d_{i,j}$	Kapazität bzw. Bitverzögerung des Ringabschnitts zw. S*i* u. S*j*	Bit

Tabelle 4.1: Im Token-Ring Beispielszenario verwendete Größen

Umsetzung sowohl hoher als auch geringer Abstraktionsniveaus.

4.4.1 Zentralisiertes Netzlast-Verhalten

Als Konzept mit hohem Emulationsgrad läßt sich das auf dem Netzlast-Verhalten basierende Konzept aus Abschnitt 4.2.2 auch zentral realisieren. Der Handel mit Kreditanteilen zwischen den verschiedenen Knoten sowie die Unterrichtung sämtlicher Stationen über die Ankunftsraten der Sendewünsche aller anderen Stationen fällt dann weg. Stattdessen formuliert ein Redirector auf jeder Station Sendewünsche für zu sendende Rahmen an den zentralen Emulationsknoten. Dieser zentrale Knoten hat nun eine globale Sicht. Alle notwendigen Daten wie bereits verbrauchte Übertragungsrate und Netzlast sind ihm bekannt und können ohne Umwege direkt in Berechnungen einfließen.

Der Emulationsknoten übernimmt eine Server-Rolle. Alle beteiligten Stationen treten als Clients auf, die Anfragen in Form von Rahmensendewünschen an den Server stellen. Bezüglich der Client-Server-Kommunikation gibt es zwei Möglichkeiten, wie Anfragen formuliert werden:

1. Clients senden über ihren Redirector vollständige Rahmen inklusive Nutzlast an den Server. Der Server liefert diese Rahmen stellvertretend entsprechend seinen Berechnungen an den bzw. die Empfänger aus.

2. Clients senden nur Sendewünsche statt vollständiger Rahmen. Als Information wird die Rahmenlänge und Client-Identifikation übermittelt. Der Server übernimmt die Rolle eines Sequenzierers und teilt in seinen Antworten den Clients mit, wie sie zu verfahren haben. Das heißt ob und wann ein Client den beantragten Rahmen direkt an den Empfänger senden darf.

Grundsätzlich wird unabhängig von der Anfrageart eine Flußsteuerung zwischen Clients und Server mit Hilfe von Bestätigungsrahmen vorgenommen. Dies dient der Rückkopplung vom Server zu den Clients über die vom Server verarbeitbare Emulationslast. Ansonsten würde ein Client, der eine sehr hohe Rate an Sendewünschen aufweist, unter Umständen ununterbrochen mit bis zu 1 GBit/s Anfragen an den Server stellen, obwohl beispielsweise ein Netz mit einer Übertragungsrate von nur 4 MBit/s emuliert werden soll. In diesem Fall würde der Client nicht nur soviel Übertragungsrate verbrauchen, wie eigentlich alle Clients am gemeinsamen Medium zusammen, sondern weit darüber hinaus, was nicht sinnvoll ist.

Für die zwei oben genannten Arten der Client-Server-Anfragen ist jeweils ein Beispielszenario in Abb. 4.13 zu sehen. Dem Beispiel könnte das Schiebefensterprotokoll zugrundeliegen, so daß – wie in der Abbildung – Bestätigungen für mehrere Rahmen ausstehen können. Die linke Teilabbildung (a)

zeigt die erste Möglichkeit, in welcher der Emulations-Server die an ihn umgeleiteten Rahmen stellvertetend für den sendenden Client an das Ziel ausliefert. Die Flußsteuerung erfolgt über Bestätigungs- bzw Ablehnungsrahmen, die der Server an den entsprechenden Client zurücksendet. In der rechten Teilabbildung (b) wird die zweite Möglichkeit dargestellt, in welcher der Emulations-Server die Rolle eines Sequenzierers übernimmt. Ein Client sendet hier nur noch kurze Anfragen an den Server und darf nach einer Bestätigung den eigentlichen Rahmen mit Nutzlast direkt an das Ziel senden.

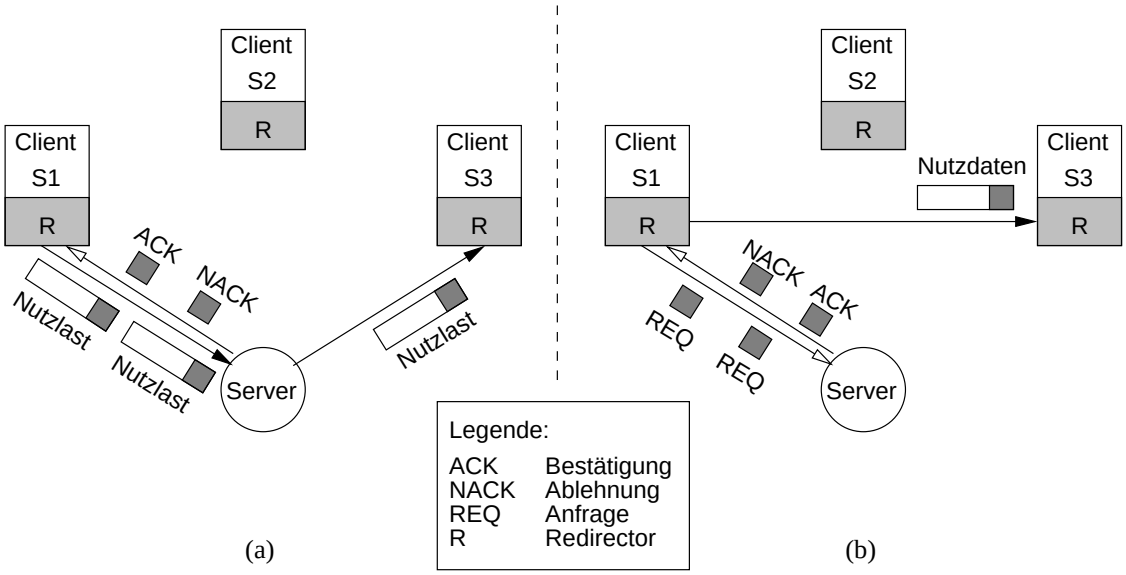


Abbildung 4.13: Zwei Schemata eines zentralisierten Emulationskonzepts

4.4.2 Zentralisierte MAC-Nachbildung

Die Konzepte mit geringem Abstraktionsniveau aus Abschnitt 4.3 lassen sich prinzipiell auch in einer zentralen Emulationsarchitektur umsetzen. Im Gegensatz dazu wird nun kein abstraktes Berechnungsmodell eingesetzt, sondern das Medienzugriffsverfahren zentralisiert in Software nachgebildet. Dabei verlagert sich quasi die Zugriffskoordination der MAC-Ebene in „das Medium“ in Form eines zentralen Emulationsknotens.

Wie im vorhergehenden Abschnitt beschrieben, ergibt sich eine Client-Server Struktur. Hier existieren dieselben zwei Möglichkeiten der Kommunikationsart zwischen Client und Server.

1. Der Server liefert Rahmen stellvertretend für Clients an das Ziel aus.

2. Der Server nimmt nur Sendewünsche entgegen und teilt dem jeweiligen Client mit, ob und wann er seinen Rahmen selbst direkt an das Ziel senden darf.

Auch hier wird wie im vorangehenden Abschnitt eine Flußsteuerung benötigt, um Clients davon abzuhalten, mit unnötig hoher und evtl. nicht mehr verarbeitbarer Rate, Anfragen an den Server zu senden.

4.5 Konzeptübersicht

Tab. 4.2 zeit eine Aufstellung der in den vorhergehenden Abschnitten vorgestellten Emulationskonzepte.

Konzept	**Netztyp**	**Merkmale**		
		Implementierung	**Arch.**	**Emulationsgrad**
Netzlast-Verhalten, 4.2.2	beliebiger	Sicherungsschicht	verteilt	abstraktes Modell
Ethernet, 4.3.1	Ethernet	Sicherungsschicht	verteilt	Nachbildung
Token-Ring, 4.3.2	Token-Ring	Sicherungsschicht	verteilt	Nachbildung
zentr. Lastverhalten, 4.4.1	beliebiger	Sicherungsschicht	zentral	abstraktes Modell
zentr. MAC-Nachbildung, 4.4.2	bestimmter	Sicherungsschicht	zentral	Nachbildung

Tabelle 4.2: Übersicht über die vorgestellten Emulationskonzepte

Kapitel 5

Evaluation

Die im vorangegangenen Kapitel 4 vorgestellten Emulationskonzepte werden im folgenden anhand der in Kapitel 3 aufgestellten Kriterien evaluiert. Die Bewertung dient dazu, ein erfolgversprechendes Konzept auszuwählen, um dieses prototypisch zu implementieren.

Für jedes der Konzepte Netzlast-Verhalten (4.2.2), Ethernet-Emulation (4.3.1), Token-Ring-Emulation (4.3.2), zentralisiertes Netzlast-Verhalten (4.4.1) und zentralisierte MAC-Nachbildung (4.4.2) werden die einzelnen Evaluationskriterien diskutiert. Die zusammenfassende Bewertung in der Übersicht erfolgt am Ende dieses Kapitels.

5.1 Netzlast-Verhalten

Das Emulationskonzept verteilt zunächst die gemeinsame Bandbreite des Netzes an die einzelnen Stationen beispielsweise mit Hilfe des Kredit-Verfahrens. Außerdem werden die von der Netzlast abhängigen Parameter Durchsatz und mittlere Verzögerungszeit bis zum erfolgreichen Medienzugriff der jeweils einer Station zugeteilten Bandbreite überlagert, indem Rahmen mit zusätzlicher Verzögerung beaufschlagt werden. Die dynamische Anpassung der Kreditanteile und das Messen der Netzlast erfolgt in möglichst kurzen Zeitabständen.

1) **Anzahl emulierter Netztypen.** Da das Konzept durch die austauschbaren Kennlinien praktisch beliebig parametrisierbar ist, können die verschiedensten Netztypen emuliert werden. Prinzipiell ist die Anzahl emulierter Netztypen nach oben offen.

2) **Realismus.** Da das Konzept prinzipiell für jeden Netztyp eingesetzt werden könnte, müßte es für ein möglichst reales Emulationsverhalten sämtliche Spezifika aller Netztypen mit gemeinsamem Medium nachbilden. Um eine endliche, handhabbare Menge von Spezifika zur Verfügung zu haben, werden die folgenden Elemente betrachtet (in Klammern sind Netztypen angegeben, die das jeweilige Merkmal aufweisen):

1. Lastabhängiger Durchsatz (alle Medienzugriffsverfahren),
2. lastabhängige durchschnittliche Medienzugriffsverzögerung (alle Medienzugriffsverfahren),
3. Trägererkennung (CSMA),
4. Kollisionserkennung (CSMA/CD),
5. Persistenz (1-, p-, non-persistentes CSMA),

6. Indeterminismus (binäres Backoff-Verfahren bei Ethernet),
7. Token (Token-Ring, Token-Bus, FDDI),
8. Determinismus (Token-Ring, Token-Bus, FDDI),
9. Prioritäten (Token-Ring, Token-Bus, FDDI, DQDB),
10. Monitorstation (Token-Ring),
11. isochroner Netzverkehr (Token-Bus, FDDI, DQDB),
12. FIFO-Warteschlange für Sendereihenfolge aller Stationen (DQDB).

Wenn also das Idealkonzept unter Umsetzung aller 12 Spezifika 100 % Realismus erreicht, so weist das betrachtete Konzept mit 2 von 12 Merkmalen, nämlich lastabhängiger Durchsatz und Verzögerung, 17 % Realismus auf.

3) **Single Point of Failure.** Alle Stationen müssen funktionsfähig sein, um am Kreditaustausch sowie der Verteilung der lokalen Ankunftsraten λ teilnehmen zu können. Der Ausfall einer einzigen Station würde das Emulationsverhalten stark stören, indem beispielsweise die ausgefallene Station all ihre Kreditanteile fälschlicherweise behält, obwohl sie nichts mehr senden kann. Hinzu kommt noch der Switch der Emulationsumgebung, bei dessen Ausfall überhaupt keine Kommunikation mehr möglich ist. Es existieren also $N+1$ Single Points of Failure bei N Knoten im System.

4) **Flaschenhals.** Die Stationen stellen keine Flaschenhälse im System dar, da sie alle gleichberechtigt an der Emulation teilnehmen und dieselben Berechnungen und Tätigkeiten ausführen. Der einzige Flaschenhals ist der Switch bzw. dessen limitierte Pufferkapazität, die bei Überlastung von Ports zu Rahmenverlust führen kann.

5) **Overhead.** Der Berechnungs-Overhead für die Abbildung von momentaner Netzlast auf Durchsatz und Verzögerung ist sehr gering im Falle eines einfachen Nachschlagens in einer Tabelle.

Der Kommunikations-Overhead für den Kreditaustausch und die Verteilung der Ankunftsraten kann jedoch prinzipiell über 100 % betragen. Dies wäre dann der Fall, wenn für jeden gesendeten Rahmen Kreditanteile getauscht und die neue, evtl. veränderte Ankunftsrate mitgeteilt wird. Mit der Kommunikation steigt natürlich auch der interne Berechnungs-Overhead, da die Nachrichten entsprechend verarbeitet werden müssen.

Diese Verarbeitung verlängert die durchschnittliche Verzögerung von Rahmen im Emulationssystem und beeinflußt somit die Echtzeitfähigkeit (Kriterium 6) auf negative Weise. Außerdem wird unter Umständen der emulierte Netzdurchsatz stärker limitiert, je mehr Kommunikationsverkehr über die Emulationstransportumgebung befördert werden muß, da dies Bandbreite kostet, die für Nutzdaten nicht mehr zur Verfügung steht. Dieser Einfluß kann möglicherweise dadurch relativiert werden, daß Emulationssteuerungskommunikation über das separate – aber langsamere – administrative Netz abgewickelt wird.

Hält man andererseits die Kommunikation und somit auch die interne Verarbeitung gering, so reagiert das System nur sehr träge auf burst-artige Sendewünsche. Gerade dieses selbstähnliche Verkehrsmuster mit Sende-Bursts tritt aber beispielsweise im praktischen Betrieb bei Ethernet auf [PF94]. Wenn nun das System träge ist, kommt die Reaktion zu spät und das Emulationsverhalten kann vom Realverhalten stark abweichen.

6) **Echtzeitfähigkeit.** Die durchschnittliche Verzögerung eines Rahmens im Emulationssystem steht und fällt mit der Häufigkeit, mit der Kredit- und Ankunftsratenaustausch, d.h. Kommunikations-Over-

head, stattfindet. Als grobe Schätzung reicht das Intervall von wenigen bis zu mindestens einigen zehn Millisekunden.

7) **Netzlast.** Unter der vereinfachenden Annahme, die Emulationssteuerungskommunikation wird über das separate, administrative Netz transportiert, steht zunächst die volle physikalische Bandbreite der Emulationsumgebung für die Nutzdaten zur Verfügung. Allerdings muß bei der Häufigkeit des Kredit- und Ankunftsratenaustausches ein Kompromiß eingegangen werden, der zu Verzerrungen im Emulationsverhalten führt. Dadurch können burst-artige Sendewünsche nur abgeflacht ausgeführt werden oder es wird unnötigerweise die Bandbreite durch veraltete Ankunftsraten limitiert. Umgekehrt kann bei einer zu optimistischen Verhaltensweise zu viel Verkehr erzeugt werden, was verworfene Rahmen im Switch zur Folge hat. Dies bringt negative Auswirkungen auf das Emulationsverhalten mit sich, da mit dem Phänomen verlorener Rahmen nicht gerechnet wird und auch nicht sinnvoll umgegangen werden kann. In beiden Fällen wird die erreichbare Last beschränkt und erreicht nicht ihren Sollwert.

Einen tatsächlichen, absoluten Wert für die erreichbare emulierte Netzlast anzugeben, gestaltet sich angesichts der möglichen Einflüsse schwierig. Es ist anzunehmen, daß das Maximum weit unter 1 GBit/s liegt.

8) **Umsetzbarkeit.** Das zur Aufteilung der Bandbreite verwendete Verfahren, z.B. der Kreditmechanismus, stellt selbst ein Medienzugriffsverfahren dar, das entsprechende Kennlinien für lastabhängigen Durchsatz sowie Medienzugriffsverzögerung aufweist. Um nun überhaupt mit Hilfe der Kennlinien des zu emulierenden Netztyps eine Emulation durchführen zu können, müssen die Kennlinien des Bandbreitenaufteilungsverfahrens bestimmte Anforderungen erfüllen:

- Die Durchsatzkennlinie muß stets größergleich derjenigen des zu emulierenden Netztyps sein. Auf diese Weise kann die Emulation durch Limitierung um den jeweiligen Differenzbetrag ihre Aufgabe erfüllen.

  Abb. 5.1 zeigt grau hinterlegt den Bereich, in dem sich die Durchsatzkennlinie eines Bandbreitenaufteilungsverfahrens befinden müßte, um slotted ALOHA zu emulieren. Die untere Begrenzung ist die Kennlinie von slotted ALOHA. Die obere Begrenzung ist durch das ideale Medienzugriffsverfahren gegeben.

- Analog verhält es sich mit der Verzögerungskennlinie der Bandbreitenaufteilung. Sie muß allerdings stets kleinergleich derjenigen des zu emulierenden Netztyps sein. Dadurch läßt sich die zu emulierende Verzögerung über den Differenzbetrag sicherstellen.

Die Implementierung des Kredithandels und Ankunftsratenaustausches unter Berücksichtigung der oben angegebenen Anforderungen an dessen Kennlinie stellt einen erheblichen Mehraufwand dar. Sie wird vor allem auch dadurch erschwert, daß innerhalb der Kernel-Umgebung viele Seiteneinflüsse, z.B. durch Synchronisation, existieren. Dies läßt eine Umsetzung gegenüber einer Programmierung im Benutzermodus sehr kompliziert werden. Insofern läßt sich dieses Konzept bezüglich seiner Umsetzbarkeit mit dem Wert 5 einstufen.

9) **Relevanz.** Da sich mit dem Konzept beliebige Netztypen inklusive Ethernet emulieren lassen, wird seine Relevanz mit 1 bewertet.

5.2 Ethernet-Emulation

Das Emulationskonzept bildet das Medienzugriffsverfahren CSMA/CD in Software auf Sicherungsschicht nach. Hierzu wird die Annahme getätigt, daß die Übermittlungszeit von Nachrichten in der Emulationsumgebung vernachlässigbar klein gegen die von der Emulation eingeführten Verzögerungen ist.

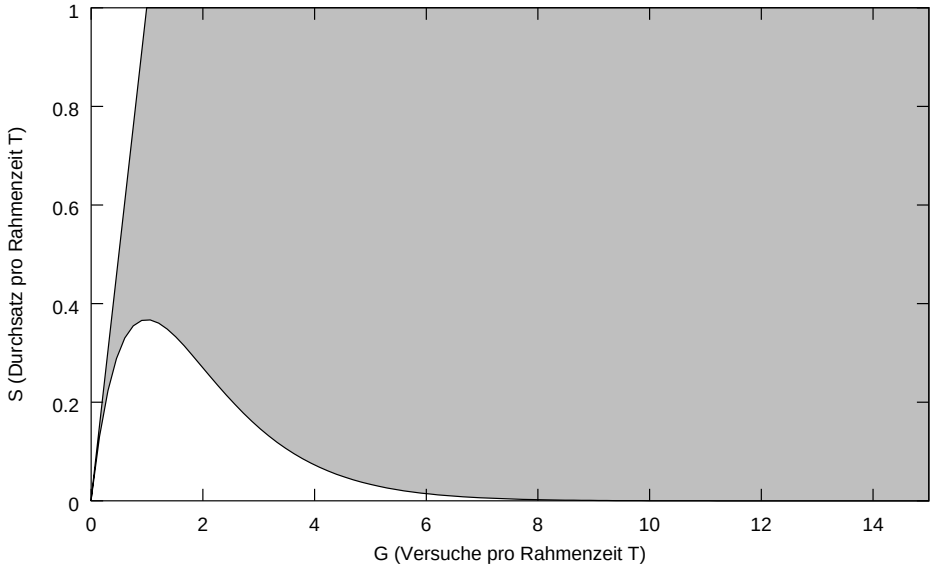


Abbildung 5.1: Anforderungen an die Durchsatzkennlinie eines Bandbreitenaufteilungsverfahrens

Anhand von Vergleichen lokaler Zeitstempel erkennen die Stationen unabhängig voneinander emulierte Signalträger und deren Überschneidungen, d.h. Kollisionen. Die Übertragungsgeschwindigkeit wird durch Verzögerung in Abhängigkeit von der Signalausbreitungszeit und der Rahmenlänge emuliert.

1) **Anzahl emulierter Netztypen.** Die Anzahl der mit diesem Konzept emulierbaren Netztypen beträgt eins, da ausschließlich ein ganz bestimmter Typ durch Nachbildung seines Medienzugriffsverfahrens emuliert wird.

2) **Realismus.** Von den Ethernet-Spezifika Trägererkennung, Kollisionserkennung, Persistenz und Indeterminismus werden durch das Konzept alle umgesetzt. Die allgemeinen Spezifika von Medienzugriffsverfahren – lastabhängiger Durchsatz sowie durchschnittliche Medienzugriffsverzögerung – werden vom Konzept durch Nachbildung des Medienzugriffsverfahrens implizit berücksichtigt. Daraus folgt, daß nach der Definition des Kriteriums 100 % Realismus erreicht werden.

3) **Single Point of Failure.** Es existiert bei Ethernet keine zentrale Koordinationsinstanz. Alle Stationen sind voneinander unabhängig. Insofern existiert nur der Switch der Emulationsumgebung als einziger Single Point of Failure.

4) **Flaschenhals.** Flaschenhälse in Form von Knoten existieren keine. Einzig Pufferwarteschlangen im Switch können durch massive Eingabeströme von mehreren Ports zu einem Flaschenhals werden, wenn die Ausgangsbandbreite des Zielports nicht mehr ausreicht. Eine weitergehende Diskussion zur Problematik verworfener Rahmen folgt bei Kriterium 7.

5) **Overhead.** Der Kommunikations-Overhead ist praktisch Null, da das Konzept nur dann Sendungen vorsieht, wenn dies auch in einem realen Ethernet der Fall wäre. Die durchzuführenden Berechnungen sind sehr kurz und einfach. Außerdem bestehen die Verwaltungsoperationen zur Verzögerung der Rahmen größtenteils aus schnellen Zeigeroperationen, bei denen Zeiger auf Rahmen in Timer-Warteschlangen eingetragen werden. Dadurch ergibt sich ein äußerst geringer Gesamt-Overhead von wenigen Prozent.

6) **Echtzeitfähigkeit.** Ein Großteil der auszuführenden Operationen bezieht sich auf die Verzögerung von Rahmen und ist diesbezüglich mit dem vorhandenen NETShaper zur Modellierung von Punkt-zu-

Punkt Verbindungen vergleichbar. In [HR02] ist für die durch die Emulation eingeführte Verzögerung eine experimentell ermitteltete Obergenze von 3 μs angegeben. Zieht man die zusätzlichen Berechnungen in diesem Konzept mit in Betracht, sollte die hier geltende Obergrenze mit dem Zwei- bis Dreifachen abschätzbar sein. Somit liegt die unvermeidliche Verzögerung voraussichtlich unter 10 μs.

7) **Netzlast.** Je höher die emulierte Übertragungsrate gewählt wird, desto weniger gilt die Annahme, daß die Übertragungszeit für Rahmen in der Emulationsumgebung vernachlässigbar ist. Sobald die Ausdehnung übertragener Rahmen, d.h. deren Sendezeit t_r, in Betracht gezogen werden muß, werden an empfangenden Stationen festgestellte Rahmenüberlappungen wie in Abb. 4.8 immer seltener. Dies ist zurückzuführen auf die Serialisierung der Rahmen, die auf einem Switchport ausgegeben werden sollen. Dem Switch ist es nur möglich, einen Rahmen nach dem anderen auf einem Port auszugeben. Kommen für denselben Empfänger mehrere Rahmen verschiedener Sender zeitlich so nah beieinander im Switch an, daß sie sich überlappen würden, werden diese zunächst zwischengepuffert und serialisiert ausgeliefert.

Im Extremfall, wenn die emulierte gleich der physikalisch maximal möglichen Übertragungsrate ist, können kaum mehr Kollisionen von der Emulation festgestellt werden. Außerdem ist dann auch keine korrekte Trägererkennung mehr möglich. Die Netzwerkkarte eines Empfängers löst eine Unterbrechung aus, sobald sie einen Rahmen vollständig empfangen hat. Zu diesem Zeitpunkt ist dann aber bereits der Emulationszeitpunkt ‚tsr' eigentlich abgelaufen. Das Verfahren degeneriert dann zu einem reinen geswitchten Ethernet mit etwas zusätzlicher Verzögerung auf den Stationen.

Erschwerend kommt hinzu, daß durch einen Broadcast für jeden gesendeten Rahmen die Wahrscheinlichkeit vom Switch verworfener Rahmen immer größer wird. Dieser Fall kann von der Emulation nicht erkannt und behandelt werden. D.h. ein Betrieb in dieser Zone, bei der mit verworfenen Rahmen gerechnet werden muß, ist zu vermeiden.

Die erreichbare Emulationslast wird mit mindestens 10 MBit/s, möglicherweise 100 MBit/s geschätzt.

8) **Umsetzbarkeit.** Bislang unbehandelt blieb die Frage, wie genau die Auflösung der Zeitmessung sein muß, um eine realitätsnahe Emulation mit diesem Konzept durchzuführen. Außer zur Einführung der Auslieferungsverzögerung wird die Zeit benötigt, um Träger- und vor allem Kollisionserkennung zu bewältigen. Eine Kollision tritt auf, sobald auch nur ein Bit zweier oder mehrerer Rahmen sich überlappen. Daraus folgt, daß die Zeitgranularität der Dauer eines Bits entsprechen muß. Die „Länge" t_B eines Bits ist der Kehrwert der Übertragungsrate b. D.h. die Zeitmessung sollte diskrete Zeitschritte von $t_B = 1/b$ erfassen können. Tab. 5.1 führt einige gängige Ethernet-Bandbreiten mit den sich daraus ergebenden Bit-Dauern auf. Vergleicht man die angegebenen Zeiten mit Zykluszeiten um 1 ns bei derzeit gängigen Prozessoren im GHz-Bereich, so fällt auf, daß bei hohen Bandbreiten – selbst unter Verwendung taktgenauer Uhren – eine exakte Zeitmessung schwer möglich ist.

Bandbreite b	**Sendedauer eines Bits** t_B
10 MBit/s	100 ns
100 MBit/s	10 ns
1 GBit/s	1 ns

Tabelle 5.1: Bit-„Länge" in Abhängigkeit von der Bandbreite

Ein wichtiges Zeitintervall ist die Dauer eines Contention Slots von 51,2 μs. Das Intervall wird benötigt, um mit Hilfe des binären Backoff-Verfahrens den Zeitpunkt für einen erneuten Sendeversuch nach einer Kollision zu ermitteln. Ein solcher Versuch wird üblicherweise unter Verwendung eines Timers eingeplant. Nun ist aber die Auflösung der Timer im Linux-Kernel abhängig von der Interrupt-Frequenz

und beträgt standardmäßig auf der Intel-Architektur 100 Hz, d.h. die Genauigkeit liegt bei 10 ms. Durch Neuübersetzen des Kernels läßt sich die Frequenz höhersetzen, wobei maximal 1 kHz, d.h. 1 ms Auflösung, ein akzeptabler Kompromiß zwischen höherer Genauigkeit und größerem Interrupt-Overhead darstellt [Gur99]. Jedoch ist 1 ms immer noch um Zehnerpotenzen größer als Bit-Dauern aus Tab. 5.1. Falls die Genauigkeit nicht ausreichend sein sollte, läßt sich eine andere Timer-Implementierung wie beispielsweise UKA-APIC-Timer aus [WPR+02] einsetzen, die taktgenaue Timer erlaubt und somit die maximal realisierbare Genauigkeit erreicht.

Aus diesen Betrachtungen heraus erfolgt die Einordnung des Konzepts bezüglich seiner Umsetzbarkeit im besten Fall mit dem Wert 1. Sollten die vorhandenen Rahmenzeitstempel mit mindestens Mikrosekunden-Genauigkeit nicht ausreichend sein, ist ein kleiner Kernel-Patch erforderlich, der die Bewertung auf 2 herabsetzt. Bei Einsatz einer anderen Timer-Implementierung entsteht Abhängigkeit von fremder Software, die nicht zum Standard-Linux-Kernel zählt, wodurch sich die Bewertung bis auf 4 verschlechtern würde.

9) **Relevanz.** Nachdem das Konzept Ethernet als den relevantesten Netztyp emuliert, wird die Relevanz mit 1 bewertet.

5.3 Token-Ring-Emulation

Das Emulationskonzept bildet Token-Passing als Medienzugriffsverfahren in Software auf Sicherungsschicht nach. Zusätzlich wird die Auslieferung von Rahmen beim Empfänger entsprechend der Netztopologie und Rahmenlänge verzögert.

1) **Anzahl emulierter Netztypen.** Die Beschreibung des Konzepts in Abschnitt 4.3.2 verwendet Token-Ring als einzigen Netztyp im Fallbeispiel. Jedoch ist je nach Ausbaustufe des Emulationskonzepts, das allgemein auf der Grundlage des Token-Passings basiert, die Emulation der ebenfalls Token-basierten Netztypen FDDI sowie Token-Bus denkbar. Die Anzahl emulierter Netztypen läßt sich also zwischen ein bis drei variieren.

2) **Realismus.** Die Spezifika der durch dieses Konzept emulierbaren Netztypen Token-Ring, FDDI und Token-Bus bestehen aus lastabhängigem Durchsatz sowie durchschnittlicher Medienzugriffsverzögerung und außerdem Determinismus, Token, Prioritäten, Monitorstation, isochroner Verkehr. Hiervon werden nach der Beschreibung in Abschnitt 4.3.2 die grundlegenden Spezifika Token-Passing und somit implizit Durchsatz, Verzögerung sowie Determinismus emuliert. Dies entspricht einem Realismus von 57 %.

Die in der Beschreibung fehlenden bzw. nicht voll unterstützten Spezifika lassen sich je nach Anforderung problemlos in das Konzept integrieren, wodurch ein Realismus von bis zu 100 % erreicht werden kann.

Am Beispiel der Monitorfunktionalität von Token-Ring lassen sich verschiedene, mögliche Ausbaustufen veranschaulichen. Im einfachsten Fall, wenn die Topologie eine ausreichende Ringbitzahl für das Token zur Verfügung stellt und weiterhin auf die Emulation von Stations- und Kabelausfällen verzichtet wird, ist eine Monitorfunktion nicht notwendig. Wird ein Monitor doch benötigt, kann es ausreichend sein, die Monitorstation in der Topologiebeschreibung vorzugeben. Dies erspart die Implementierung des Wahlverfahrens für den Monitor. Durch die Möglichkeit der Beschreibung dynamischer Szenarien mit Hilfe der Topologiedefinitionssprache aus [HMLR01] ist sogar ein Monitorwechsel auf eine andere Station möglich. Eine vollständige Implementierung der Monitor- und zugehöriger Funktionalität ist also erst dann notwendig, wenn auf ein exaktes Verhalten ohne Vereinfachungen nicht verzichtet werden

kann.

3) **Single Point of Failure.** Die Anzahl der Single Points of Failure in diesem Konzept ist ebensogroß wie z.B. bei einem realen Token-Ring Netz. Der Ausfall einer einzigen Station unterbricht bzw. stört die Kommunikation auf dem Ring, d.h. es würden $N+1$ Single Points of Failure bei N Stationen und dem Switch der Emulationsumgebung vorliegen. Nimmt man in der Emulation eine sternförmige Verkabelung über ein Wiring Center sowie kein byzantinisches Verhalten der Stationen an, reduziert sich die Anzahl auf das Wiring Center, repräsentiert durch den Switch, als einzigen Single Point of Failure.

4) **Flaschenhals.** Es existieren keine Flaschenhälse im System.

5) **Overhead.** Der Overhead ist mit der zuvor bewerteten Ethernet-Emulation vergleichbar und beträgt wenige Prozent.

6) **Echtzeitfähigkeit.** Die durch die Emulation eingeführte durchschnittliche Verzögerung pro zu verarbeitendem Rahmen ist ebenfalls mit zuvor genannter Ethernet-Emulation vergleichbar und liegt unter 10 μs.

7) **Netzlast.** Angesichts des niedrigen Overheads ist eine nahezu vollständige Auslastung möglich. Vollast wird erreicht, wenn jede Station jeweils für ihre gesamte Token-Haltezeit sendet.

8) **Umsetzbarkeit.** Im Gegensatz zum vorhergehenden Konzept werden bei dem hier betrachteten Timer nur zur Auslieferungsverzögerung benötigt. Hierbei ist die Genauigkeit weniger kritisch und die vorhandenen Rahmenzeitstempel sowie Timer sollten ausreichend sein. Evtl. werden Rahmen durch bündelweise Bearbeitung im Timer teilweise zu spät ausgeliefert, was aber über die Zeit gemittelt keinen Fehler einführt. Somit ist eine Umsetzung in Form eines Moduls für den Standard-Linux-Kernel möglich und es ergibt sich eine Umsetzbarkeitsbewertung von 1.

9) **Relevanz.** Token-Ring findet heute immer weniger Verbreitung in LANs. Aufgrund der daraus resultierenden geringeren Bedeutung für den Praxiseinsatz wird die Relevanz des Konzepts mit 2 bewertet.

5.4 Zentralisiertes Netzlast-Verhalten

1) **Anzahl emulierter Netztypen.** Es handelt sich bei diesem Konzept um dasselbe Emulationsmodell wie in Abschnitt 5.1. Der Unterschied besteht nur in der Architektur. Insofern ist die Anzahl emulierbarer Netztypen, die durch das Modell vorgegeben wird, ebenfalls praktisch beliebig.

2) **Realismus.** Auch der Grad des Realismus wird vom Modell beeinflußt und wurde bereits mit 17 % bestimmt.

3) **Single Point of Failure.** Im Gegensatz zur verteilten Variante aus Abschnitt 5.1 fallen bei dem hier betrachteten Konzept die N Stationen als Single Points of Failure weg, da kein Kredit- und Ankunftsratenaustausch mehr stattfindet. Es können von ausgefallenen Stationen also keine Kreditanteile fälschlicherweise einbehalten werden. Single Points of Failure stellen nun die notwendigen, zentralen Emulationsknoten und der Switch der Emulationsumgebung dar. So ergeben sich insgesamt $X+1$ Single Points of Failure.

4) **Flaschenhals.** Flaschenhälse existieren in Form der notwendigen X zentralen Emulationsknoten, über die sämtliche Entscheidungen bezüglich Rahmensendewünschen laufen müssen.

5) **Overhead.** Der Kommunikations-Overhead wird sicherlich maßgeblich von der Art der Client-Anfragen bei Sendewünschen bestimmt. Zunächst ist anzunehmen, daß ein Ansatz, in dem der zentrale Emulationsknoten eine Sequenziererrolle übernimmt, weniger Overhead erzeugt, da nichterfüllbare Sendewünsche nicht unnötigerweise samt Nutzlast an den Server gesendet werden.

Angesichts der eingesetzten Emulationsumgebung kann sich diese Annahme aber durchaus relativieren. Es kommt Ethernet mit einer hohen Übertragungsrate von 1 GBit/s als Kommunikationsmedium zum Einsatz. Unter anderem um einen hohen Duchsatz zu erzielen, wird die minimale Rahmenlänge bei höheren Raten gegenüber dem ursprünglichen Ethernet vergrößert. Auf der anderen Seite soll möglicherweise ein Netz mit einer relativ geringen Übertragungsrate emuliert werden. Dann ist es möglich, daß Nutzlast aus der Emulation im minimalen Rahmen der Emulationsumgebung Platz findet. Dies bedeutet, daß statt einer Sendewunschanfrage mit Antwort, die beide größtenteils nur Füller enthalten, und anschließendem Senden des vollständigen Rahmens durch den Client nur noch ein vollständiger Rahmen an den Server und von dort an den entsprechenden Empfänger gesendet zu werden braucht. Dadurch reduziert sich die Kommunikation auf zwei Drittel. In einem solchen Fall wäre eine Server-Rolle, in welcher der Server vollständige Rahmen stellvertretend für Clients ausliefert, einer Sequenziererrolle im Sinne eines minimierten Kommunikations-Overheads vorzuziehen.

Im günstigsten Fall beträgt der Kommunikations-Overhead 100 %. Dies ist der Fall, wenn ein vollständiger Rahmen an den Server gesendet und von dort erfolgreich weitergesendet wird. Für Rahmen, die nicht erfolgreich weitergeleitet werden können, beispielsweise durch Beteiligung an einer Kollision, erhöht sich der Overhead durch ablehnende Bestätigungsrahmen, die vom Server an den ursprünglichen Sende-Client geschickt werden.

Bei Verfolgung eines Sequenzierungsansatzes wird im besten Fall ein kurzer Sendewunsch mit kurzer Antwort gesendet und je nach Antwort evtl. der Rahmen selbst. Dies verursacht nur wenige Prozent Overhead für die zwei kurzen Nachrichten, da das Senden des Rahmens selbst kein Zusatzaufwand darstellt. Im schlechtesten Falle wird davon ausgegangen, daß Nutzlastrahmen wie weiter oben beschrieben bereits vollständig in einen minimal großen Rahmen des Emulationsnetzes passen. Außerdem werden drei Nachrichten benötigt: Der Sendewunsch des Clients an den Server, daraufhin eine Bestätigung vom Server und schließlich der Rahmen selbst mit Nutzlast. Letzterer wird nicht als Overhead gerechnet, da er auch in einer Realumgebung transportiert werden müßte. Darüber hinaus bleiben zwei Rahmen, die mindestens so groß wie der Nutzlastrahmen sind, an Kommunikation. Somit entstehen 200 % Overhead bzw. entsprechend mehr, falls die minimale Rahmenlänge in der Emulationsumgebung sehr viel größer als die der Nutzlast ist.

Hinzu kommt in allen Fällen der Overhead für Modellberechnungen und evtl. die Verarbeitung von Sequenzierungsantworten auf den Clients. Da es sich um einfache, schnell auszuführende Operationen handelt, beläuft sich der Zusatzaufwand auf wenige Prozent.

6) **Echtzeitfähigkeit.** Im Gegensatz zur verteilten Variante dieses Konzeptes aus Abschnitt 5.1 fällt der Austausch von Krediten und Ankunftsraten in Form von Nachrichten über das Netz weg. Alle notwendigen Modellberechnungsdaten liegen stets aktuell auf dem zentralen Emulationsknoten vor. Somit entfällt die sonst für den Austausch notwendige Zeit. Die durchschnittliche Verzögerung pro zu verarbeitendem Rahmen ergibt sich aus wenigen Mikrosekunden für die zentrale Modellberechnung und zusätzlicher Zeit, die das Senden von Anfragen und Antworten bzw. die Rahmenweiterleitung betrifft. Bei 1 GBit/s Ethernet in der Emulationsumgebung und einer großzügig angenommenen maximalen Rahmengröße (MTU, maximum transfer unit) von 150.000 Bytes sowie einer maximalen Ausbreitungszeit von $\tau = 51,2\ \mu$s läßt sich die Obergrenze für die Übertragungszeit eines Rahmens mit $\approx 1,3$ ms abschätzen. Insgesamt liegt die unvermeidliche Verzögerung somit unter 3 ms.

7) **Netzlast.** Die beteiligten Stationen können so viel Last erzeugen, wie ihnen von der Flußkontrolle mit Hilfe der Bestätigungsrahmen erlaubt wird. Dies bedeutet, daß eine Vollast möglich ist, die nur um die von den Bestätigungsrahmen erzeugte Teillast reduziert ist.

8) **Umsetzbarkeit.** Analog zur verteilten Variante des Konzepts erfordert die Umsetzung hier einen deutlich erhöhten Mehraufwand. Zwar fällt der Kredit- und Ankunfsratenaustausch weg, jedoch müssen

Bestätigungsrahmen möglichst schnell und effizient erzeugt und verarbeitet werden. Dies bedeutet wiederum eine aufwendige Implementierung innerhalb der Kernel-Umgebung. Aus diesen Gründen wird auch dieses Konzept mit 5 bewertet.

9) **Relevanz.** Ebenso wie die verteilte Variante beinhaltet dieses Konzept die Emulation verschiedenster Netztypen inklusive Ethernet und wird somit ebenfalls mit 1 bezüglich seiner Relevanz bewertet.

5.5 Zentralisierte MAC-Nachbildung

1) **Anzahl emulierter Netztypen.** Analog zu Abschnitt 5.2 bzw. 5.3 ist die mögliche Anzahl emulierbarer Netztypen auf einen bestimmten bzw. eine kleine Menge eng verwandter Typen beschränkt.

2) **Realismus.** Wie in Abschnitt 5.3 erläutert, ist der Grad an Realismus teilweise durch verschiedene Ausbaustufen des Konzepts variierbar und kann bis 100 % betragen.

3) **Single Point of Failure.** Zur Anzahl der Single Points of Failure aus den verteilten Varianten dieses Konzepts addiert sich jeweils die Anzahl notwendiger zentraler Emulationsknoten X. Es ergeben sich beispielsweise $X+1$ Single Points of Failure für die Emulation von Ethernet bzw. $N+X+1$ für die Emulation von Token-Ring ohne Verwendung einer Sternverkabelung mit Hilfe eines Wiring Centers.

4) **Flaschenhals.** Die Anzahl der Flaschenhälse entspricht analog zu Abschnitt 5.4 der Anzahl notwendiger zentraler Emulationsknoten X.

5) **Overhead.** Für den bei diesem Konzept anfallenden Overhead ist die Diskussion aus Abschnitt 5.4 ebenfalls gültig.

6) **Echtzeitfähigkeit.** Auch bezüglich der durchschnittlichen Verzögerung pro zu verarbeitendem Rahmen gelten die entsprechenden Angaben aus Abschnitt 5.4.

7) **Netzlast.** Es gelten die Aussagen des vorangehenden Konzepts.

8) **Umsetzbarkeit.** Es gelten die Aussagen des vorangehenden Konzepts.

9) **Relevanz.** Zwar ist ein bestimmter Netztyp bei diesem Konzept nicht vorgegeben, jedoch ist die Emulation von Ethernet grundsätzlich möglich, so daß die Relevanz mit 1 bewertet werden kann.

5.6 Übersicht über die Evaluationsergebnisse

Tab. 5.2 faßt die Bewertungsergebnisse der vorangegangenen Diskussion zusammen.

In Ermangelung einer Relativskala für manche der Kriterien wird für jedes einzelne Kriterium eine Plazierung der Konzepte vorgenommen. Sie beginnt mit 1 für das, in dieser Kategorie, beste Konzept und geht bis 5 für das schlechteste. Sind mehrere Konzepte gleichwertig, erhalten sie die gleiche Plazierung, jedoch erhöht sich die Platznummer für das Folgekonzept mit jedem Konzept, das auf eine gleiche Plazierung eingestuft wird.

Üblicherweise würde die Gesamtbewertung eines Konzepts durch ein gewichtetes Mittel berechnet werden. Eine Gewichtung für die aufgestellten Kriterien scheint jedoch ausschließlich subjektiv möglich. Zudem handelt es sich bei den einzelnen Kriterienbewertungen um Plazierungen und keine absoluten Bewertungen. Um in dieser Hinsicht neutral zu bleiben und keine weiteren, willkürlichen Beeinflussungen des Gesamtergebnisses zu ermöglichen, wird auf eine Gewichtung verzichtet. Stattdessen werden die Plazierungen eines Konzepts aufaddiert und durch die Anzahl der Kriterien (9) geteilt, um den arithmetischen Mittelwert zu erhalten. Das Ergebnis der Bewertung ist in Tab. 5.3 dargestellt und erlaubt den direkten Vergleich der Emulationskonzepte.

Aus der Gesamtbewertung geht hervor, daß die Ethernet-Emulation, wie sie in Abschnitt 4.3.1 vorgestellt wurde, knapp die beste Bewertung erhält. Als erfolgversprechendes Konzept wird dieses ausgewählt und im weiteren näher betrachtet.

Kriterien	Konzepte				
	Netzlast-Verhalten (5.1)	**Ethernet-Emulation (5.2)**	**Token-Ring-Emulation (5.3)**	**zentr. Last-verhalten (5.4)**	**zentr. MAC-Nachbildung (5.5)**
# Netztypen	beliebig	1	1 – 3	beliebig	wenige
Realismus	17 %	100 %	57 % – 100 %	17 %	bis 100 %
Single Point of F.	$N+1$	1	1 bzw. $N+1$	$X+1$	$X+1$ bis $N+X+1$
Flaschenhals	1	1	0	X	X
Overhead	bis 100 %	wenige %	wenige %	100 % – $>$ 200 %	100 % – $>$ 200 %
Echtzeitfähigkeit	$< a \cdot 10$ ms	< 10 μs	< 10 μs	$<$ 3 ms	$<$ 3 ms
Netzlast	$\ll$ 1 GBit/s	$>$ 10 MBit/s	$\approx$ 1 GBit/s	nahe 1 GBit/s	nahe 1 GBit/s
Umsetzbarkeit	5	1, 2 bzw. 4	1	5	5
Relevanz	1	1	2	1	1

Tabelle 5.2: Übersicht über die Evaluation der vorgestellten Emulationskonzepte

Kriterien	Konzepte				
	5.1	**5.2**	**5.3**	**5.4**	**5.5**
# Netztypen	1	5	3	1	4
Realismus	4	1	3	4	2
Single Point of F.	4	1	3	2	5
Flaschenhals	2	2	1	4	4
Overhead	3	1	1	4	4
Echtzeitfähigkeit	5	1	1	3	3
Netzlast	4	4	1	2	2
Umsetzbarkeit	3	2	1	3	3
Relevanz	1	1	5	1	1
Summe	27	18	19	24	28
Gesamtbewertung	**3.0**	**2.0**	**2.1**	**2.7**	**3.1**

Tabelle 5.3: Abschließende Bewertung

Kapitel 6

Ausgewähltes Konzept „Ethemu“

Als erfolgversprechendes Konzept wurde in der vorangehenden Evaluation die Ethernet-Emulation nach der Beschreibung in Abschnitt 4.3.1 ausgewählt. Als Kurzbezeichnung des Konzepts wird der Name „Ethemu“ eingeführt. In diesem Kapitel werden für Ethemu Spezifikation, Systementwurf, prototypische Implementierung sowie Ergebnisse der Testläufe vorgestellt.

6.1 Spezifikation

Im folgenden wird die Spezifikation der Ethernet-Emulation festgelegt. Sie beschreibt die externen Programmier- und Konfigurationsschnittstellen. Die Spezifikation dient als Grundlage für den nachfolgenden Systementwurf.

6.1.1 Grundlagen

Das Emulationskonzept soll als Linux-Kernel-Modul implementiert werden. Als Kernel-Version wird der derzeit stabile Zweig 2.4 eingesetzt. Entsprechend der Hardware-Ausstattung der NET-Emulationsumgebung ist eine Systemplattform-Beschränkung auf Intel-Prozessoren ab Pentium aufwärts in Einprozessor-Konfiguration zulässig.

Die Programmierung erfolgt in der Programmiersprache C. Aus der Kernel-Dokumentation werden [Tor01] bezüglich des Kodierstils und [Wau01] bezüglich der Programmdokumentation als Kodierrichtlinien herangezogen. Dies soll die nahtlose Integration mit dem Linux-Kernel-Quelltext sicherstellen.

Die Quelltexte werden, wie der Linux-Kernel selbst, unter der GNU GPL (General Public License) veröffentlicht.

Kommentare, Bezeichner, Quelltextdokumentation und sonstige natürlichsprachliche Zeichenketten im Quelltext werden in englischer Sprache verfaßt. Zusammen mit der Veröffentlichung unter der GNU GPL ermöglicht dies die weltweite Weiterentwicklung der Implementierung über das Internet, dessen Standardsprache Englisch ist.

6.1.2 Modulaktivierung und -deaktivierung

Linux-Kernel-Module werden mit Hilfe des Standard-Benutzerprogrammes **insmod** dynamisch zur Laufzeit zum Kernel hinzugeladen. Hierbei lassen sich Startparameter an das zu ladende Modul übergeben. Die Aktivierung des Ethernet-Emulations-Moduls ohne Parameter geschieht mit folgendem Befehl unter einem Benutzer mit ausreichenden Systemrechten (z.B. root), wobei *ethemu* der Name des Moduls ist:

insmod ethemu

Da eine Station in einer zu emulierenden Netztopologie durchaus an mehreren, verschiedenen LANs angeschlossen sein kann, wird der Startparameter *devices* zu Verfügung gestellt. Er ermöglicht die Angabe, wieviele Ethernet-Geräte für die Emulation zu Verfügung gestellt werden sollen. Der Parameter wird mit Leerzeichen getrennt nach dem Modulnamen in der Form „devices=*n*“, mit *n* für die Geräteanzahl, angegeben. Eine Beispielmodulaktivierung mit Unterstützung für 3 Geräte lautet wie folgt:

insmod ethemu devices=3

Nach erfolgreicher Aktivierung steht eine entsprechende Anzahl an Ethernet-Emulations-Netzwerkgeräten zur Verfügung, die mit dem Standard-Benutzerprogramm **ifconfig**, wie jedes andere Netzwerkgerät auch, eingestellt werden. Die Gerätenamen setzen sich zusammen aus der Zeichenkette „ethemu“, gefolgt von einer natürlichen Zahl im Intervall $[0, n-1]$. Sie werden ausgehend von Null durchnumeriert.

Vor der Verwendung der Netzwerkgeräte müssen diese jedoch für die Emulation speziell konfiguriert werden. Dies wird im folgenden Abschnitt 6.1.3 beschreiben.

Nach Gebrauch lassen sich die Emulationsgeräte auch wieder aus dem Kernel entfernen und geben somit den von ihnen belegten Speicher und evtl. andere Ressourcen wieder frei. Vor dem Entfernen müssen alle Emulationsgeräte geschlossen sein. Anschließend kann das Entladen des Moduls mit dem folgenden Befehl ausgeführt werden:

rmmod ethemu

6.1.3 Konfiguration

Zur Konfiguration von Ethemu wird eine low-level Programmierschnittstelle sowie eine high-level Schnittstelle zur einfachen und unkomplizierten Benutzung aus einer praktisch beliebigen Umgebung heraus zur Verfügung gestellt. Diese beiden Schnittstellen werden nachfolgend definiert.

Low-level Schnittstelle

Die Konfiguration der Ethernet-Emulations-Geräte erfolgt mit Hilfe des Systemrufes ioctl (I/O ConTroL), der zur Steuerung von Ein-/Ausgabegeräten dient. Hierzu wird das ioctl-Kommando SIOCDEVPRIVATE für Netz-Sockets in Verbindung mit dem Selektor „ifr_data“ der Datenstruktur „struct ifreq“ (aus <linux/if.h>) verwendet. Über diesen Selektor wird die eigene Konfigurations-Datenstruktur „struct ethemu_conf“ übergeben. Die Struktur ist wie folgt definiert:

```
#include <linux/if.h> /* IFNAMSIZ */
struct ethemu_conf {
  enum ethemu_conf_cmds eec_cmd; /* Kommando bestimmt eec_u-Union-Selektor */
  union {
    char          eecu_name[IFNAMSIZ]; /* Name eines realen Ethernet-Geräts */
    __u32         eecu_speed;          /* Übertragungsrate in Bit/s */
    __u32         eecu_contslottime ;  /* Contentionslot -Dauer in ns */
    __u32         eecu_jambittime;     /* Jambit-Dauer in ns */
    __u32         eecu_minframesize;   /* minimale Rahmengröße in Bytes */
    struct nodelink eecu_nodelink;     /* Signalausbreitungszeit */
  } eec_u;
};
```

Für den Selektor „eec_cmd“ sind die folgenden Kommandos definiert:

ETHEMU_SET_DEV Bindet das Ethernet-Emulations-Gerät an ein reales Gerät, über das letztendlich die Rahmen übertragen bzw. von ihm empfangen werden. Das Gerät ist üblicherweise ein

bestimmtes VLAN-Pseudogerät, mit dessen Hilfe die LAN-Zugehörigkeit und Topologie festgelegt wird. Im Selektor eecu_name wird das reale Netzgerät spezifiziert.

ETHEMU_DEL_DEV Hebt die Bindung des Ethernet-Emulations-Gerätes an ein reales Gerät auf. Es werden keine weiteren Parameter verwendet.

ETHEMU_GET_DEV Liefert den Namen des realen Gerätes, an welches das Ethernet-Emulations-Gerät gebunden ist. Das Ergebnis wird im Selektor eecu_name zurückgeliefert.

ETHEMU_SET_SPEED Setzt die Übertragungsrate des emulierten Netzes. Die Rate wird in Bit/s im Selektor eecu_speed angegeben.

ETHEMU_GET_SPEED Liefert die Übertragungsrate des emulierten Netzes in Bit/s im Selektor eecu_speed zurück.

ETHEMU_SET_CONTSLOTTIME Setzt die Contentionslot-Dauer des emulierten Netzes. Die Dauer wird in Nanosekunden im Selektor eecu_contslottime übergeben.

ETHEMU_GET_CONTSLOTTIME Liefert die Contentionslot-Dauer des emulierten Netzes in Nanosekunden im Selektor eecu_contslottime zurück.

ETHEMU_SET_JAMBITTIME Setzt die Jambit-Dauer des emulierten Netzes. Die Dauer wird in Nanosekunden im Selektor eecu_jambittime übergeben.

ETHEMU_GET_JAMBITTIME Liefert die Jambit-Dauer des emulierten Netzes in Nanosekunden im Selektor eecu_jambittime zurück.

ETHEMU_SET_MINFRAMESIZE Setzt die minimale Rahmengröße des emulierten Netzes. Die Größe wird in Bytes im Selektor eecu_minframesize spezifiziert.

ETHEMU_GET_MINFRAMESIZE Liefert die minimale Rahmengröße des emulierten Netzes in Bytes im Selektor eecu_minframesize zurück.

ETHEMU_ADD_NODELINK Macht die Signalausbreitungsdauer zwischen einer bestimmten fremden Stationen und dieser Station bekannt bzw. überschreibt den vorherigen Wert.

ETHEMU_DEL_NODELINK Entfernt den Eintrag für die Signalausbreitungsdauer zwischen einer bestimmten fremden Stationen und dieser Station.

In „ethemu_conf“ wird eine weitere Struktur „struct nodelink“ zur Definition der Topologieverzögerung t_s (siehe Gleichung 4.1) zwischen einer fremden Station und der zu konfigurierenden verwendet:

```
#include <linux/if_ether.h> /* ETH_ALEN */
struct nodelink {
  unsigned char nl_hwaddr[ETH_ALEN]; /* MAC-Adresse der fremden Station */
  __u32         nl_t_s ;             /* Signalausbreitungszeit in ns */
};
```

Für die Kommandos ETHEMU_ADD_NODELINK und ETHEMU_DEL_NODELINK wird jeweils die fremde Station anhand ihrer MAC-Adresse im Selektor nl_hwaddr übergeben. Im Falle von ETHEMU_ADD_NODELINK muß außerdem die zugehörige Signalausbreitungsdauer in Nanosekunden im Selektor nl_t_s gesetzt werden.

Falls auf einem Ethernet-Emulations-Gerät ein Rahmen empfangen wird, für dessen Quell-MAC-Adresse keine Signalausbreitungszeit konfiguriert wurde, dann wird eine Kernel-Fehlermeldung ausgegeben und der Rahmen ignoriert.

High-level Schnittstelle

Für die einfache Konfiguration aus Skripten o.ä. wird das Hilfsprogramm **eeconfig** zur Verfügung gestellt. Es ermöglicht mit Hilfe von Kommandozeilenparametern die Einstellung der Ethernet-Emulations-Geräte. Eeconfig benutzt intern wiederum die im vorangehenden Abschnitt beschriebene low-level Schnittstelle. Das Programm erwartet als ersten Parameter den Emulations-Gerätenamen „ethemu*n*“ und unterstützt im Anschluß daran die folgenden Optionen jeweils mit nachfolgendem Wert, der anstelle der Bezeichner in spitzen Klammern anzugeben ist:

–d <realer Gerätename> : Siehe Beschreibung des Kommandos ETHEMU_SET_DEV.

–u : Siehe Beschreibung des Kommandos ETHEMU_DEL_DEV.

–b <Übertragungsrate> : Siehe Beschreibung des Kommandos ETHEMU_SET_SPEED.

–c <Contentionslot-Dauer> : Siehe Beschreibung des Kommandos ETHEMU_SET_CONTSLOTTIME.

–j <Jambit-Dauer> : Siehe Beschreibung des Kommandos ETHEMU_SET_JAMBITTIME.

–m <minimale Rahmengröße> : Siehe Beschreibung des Kommandos ETHEMU_SET_MINFRAMESIZE.

–a <MAC-Adresse>,<Signalausbreitungszeit> : Macht die Signalausbreitungsdauer zwischen der fremden Station mit der angegebenen MAC-Adresse und dieser Station bekannt bzw. überschreibt den vorherigen Wert. Diese Option kann mehrfach angegeben werden, um mehrere Signalausbreitungsdauern auf einmal einzustellen. Die MAC-Adresse wird in der üblichen Form als Folge von sechs hexadezimalen Bytes, die durch Doppelpunkte abgetrennt sind, erwartet. Die Signalausbreitungsdauer in Nanosekunden muß eine dezimale Zahl sein.

–r <MAC-Adresse> : Entfernt einen Eintrag für die Signalausbreitungsdauer zwischen der fremden Stationen mit der angegebenen MAC-Adresse und dieser Station. Diese Option kann mehrfach angegeben werden, um mehrere Signalausbreitungsdauern auf einmal zu entfernen.

–l : Gibt alle momentan eingestellten Parameter in lesbarer Form aus.

6.1.4 Algorithmus

Die Funktionsweise der Ethernet-Emulation wurde bereits in Abschnitt 4.3.1 umgangssprachlich beschrieben. Um einen Gesamtüberblick über die Funktionsweise zu ermöglichen, zeigt Abb. 6.1 Datenflüsse empfangener und zu sendender Rahmen sowie die zentralen Kontrollflüsse und deren Seiteneffekte.

Innerhalb der Kontrollflüsse sind die verschiedenen Entscheidungen, die anhand von Zeitstempeln getroffen werden, abgebildet. In schräggestellter Schrift sind die jeweiligen natürlichsprachlichen Interpretationen der Entscheidungsausgänge dargestellt. Auf eine Darstellung der Aktualisierung der lokalen Variablen *ts, tsr, kollision, sendend, aktiv* und *kollisionszahl* wurde der Übersichtlichkeit halber verzichtet.

Es ist zu erkennen, daß der Datenfluß hauptsächlich aus dem Einreihen in bzw. Entnehmen aus jeweils einer Sende- und einer Empfangswarteschlange besteht. Die abgebildeten Warteschlangen existieren intern in einem Ethemu-Gerät und sind von außen nicht sicht- oder veränderbar. Diese internen Warteschlangen dienen nur der Umsetzung von Verzögerung in der Emulation und sind nicht zu verwechseln mit den standardmäßig im Kernel vorhandenen Warteschlangen der Linux-Netzimplementierung.

Der zeitliche Ablauf wird von zwei Timern gesteuert. Der Sende-Timer löst erstmalige Sendeversuche inklusive Warten auf ein eventuelles Trägerende und Sendewiederholungen aus. Der Empfangs-Timer hat nur eine Aufgabe, nämlich das Ausliefern korrekt empfangener Rahmen an die Vermittlungsschicht. Die für Ethernet typische Kollisionserkennung, d.h. das Abhören des Mediums während des eigenen Sendevorgangs, ist auf Empfangsseite mitenthalten. Falls eine Kollision absehbar und die Station aktiv daran beteiligt ist, wird das Senden abgebrochen (nicht dargestellt) und ein erneuter Sendeversuch über den Sende-Timer eingeplant.

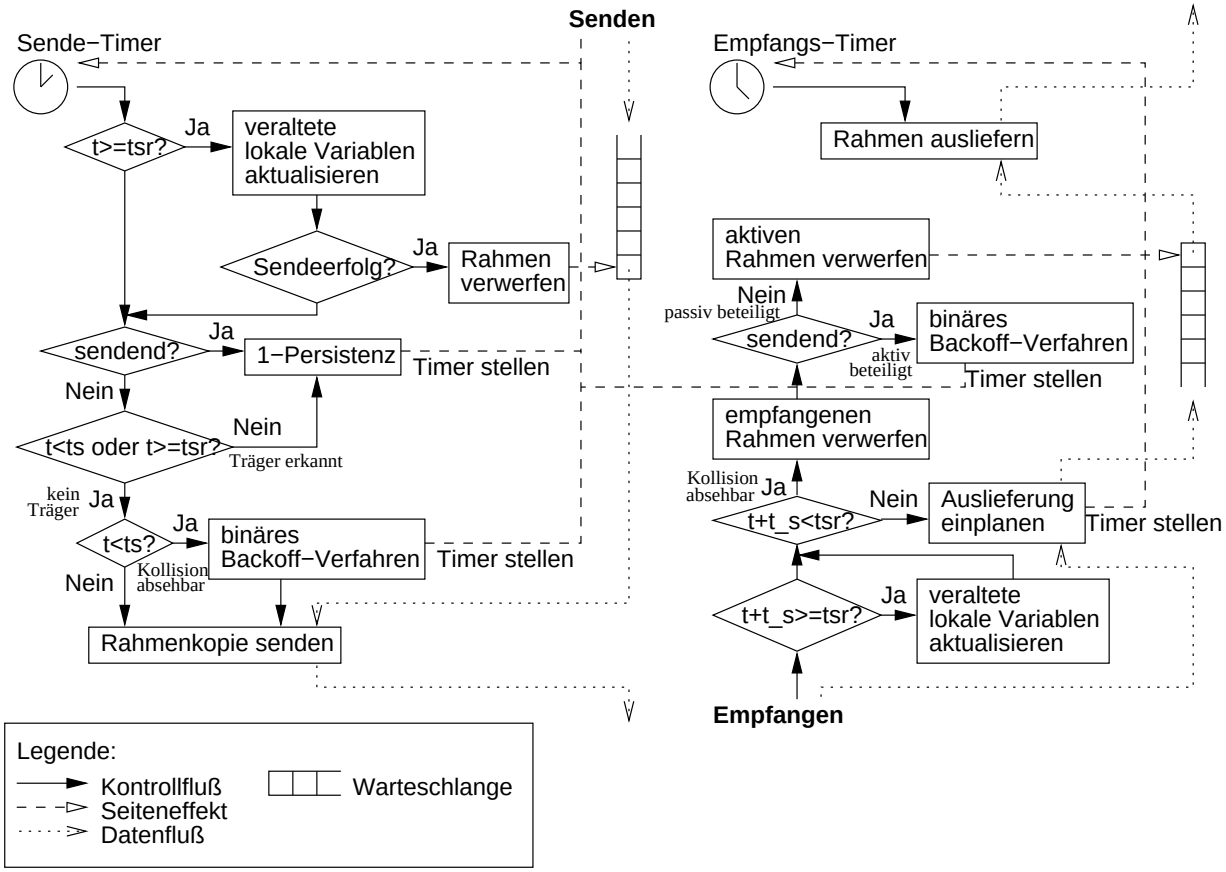


Abbildung 6.1: Schematische Funktionsweise des Ethernet-Emulations-Algorithmus

6.2 Systementwurf

Ausgehend von der Spezifikation in Abschnitt 6.1 wird im folgenden der Systementwurf für das Ethernet-Emulations-Konzept vorgestellt. Er vervollständigt die Beschreibung der vorangehenden Spezifikation durch die Darstellung der internen Zusammenhänge. Der Entwurf ist in mehrere Teile gegliedert, die sich aus Integration in den Kernel, Modulaktivierung, Konfiguration und einer Erläuterung der Algorithmenstruktur zusammensetzen. Die jeweiligen Darstellungen beschränken sich auf die wichtigsten Funktionsprototypen und Aspekte, die zum Verständnis der Funktionsweise notwendig sind. Eine vollständige Programmierschnittstellenbeschreibung befindet sich in speziellen Programm-Code-Kommentaren, aus denen sich automatisiert verschiedene Ausgabeformate wie beispielsweise DocBook, HTML,

Text oder Postscript generieren lassen [Wau01].

6.2.1 Integration in den Linux-Kernel

Das Ethemu-Modul integriert sich wie in Abb. 6.2 dargestellt in den Protokollstapel des Linux-Kernels. Im Sendefall empfängt es zu sendende Rahmen aus der Vermittlungsschicht wie ein gewöhnlicher Netzwerkgerätetreiber auf Sicherungsschicht. Nach entsprechender Behandlung eines solchen Rahmens wird er entweder verworfen oder nimmt zu gegebener Zeit seinen Weg über das angebundene reale Netzwerkgerät zum Empfänger. Im Empfangsfall paßt das Ethemu-Modul empfangene Rahmen ab, bevor sie die Vermittlungsschicht erreichen können. Auch hier wird ein solcher Rahmen wiederum entweder verworfen oder zu gegebener Zeit an die endgültige Vermittlungsschichtimplementierung ausgeliefert.

Der VLAN-Gerätetreiber innerhalb der gestrichelten Umrahmung ist prinzipiell optional. Zwar benötigt das Ethemu-Modul selbst diesen nicht, jedoch dürfte der Treiber zur Bildung von Topologien wie in Abschnitt 2.3 beschrieben in der NET-Umgebung permanent präsent sein.

Durch den Entwurf des Protokollstapels von Linux erscheinen Ethemu bzw. VLAN im Empfangspfad auf Vermittlungsschicht, obwohl es sich eigentlich um Gerätetreiber auf Sicherungsschicht handelt. Das ist jedoch nur die Implementierungssicht. In der logischen Sichtweise arbeiten die Ethemu- wie auch die VLAN-Implementierung ausschließlich auf Sicherungsschicht. Die Abweichung in der Entwurfsstruktur resultiert daraus, daß offenbar nur auf diese Weise empfangene Rahmen von realen Netzwerkgerätetreibern abgefangen werden können. Hierzu geben sich die Datensenken als Vermittlungsschichtimplementierung aus. Im Gegensatz zu letzteren liefert aber die Ethemu- bzw. VLAN-Implementierung ihre Rahmen nicht an die Transportschicht aus, sondern speist sie wie frisch empfangene Rahmen auf Sicherungsschicht wieder ein. Details hierzu folgen in Abschnitt 6.2.2.

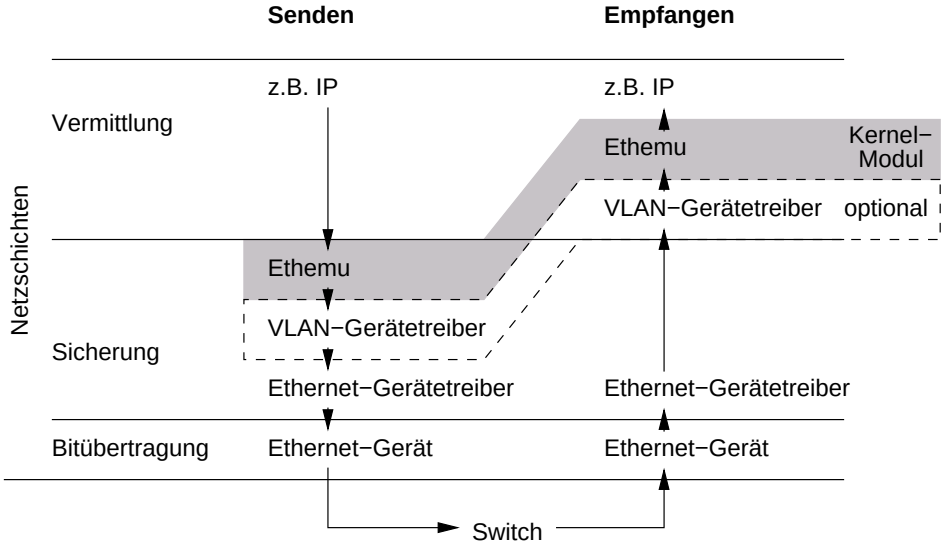


Abbildung 6.2: Schichtenarchitektur der Linux-Netzimplementierung

6.2.2 Modulaktivierung und -deaktivierung

Ein Linux-Kernel-Modul weist verschiedene spezielle Bezeichner, Abschnitte im Objektkode und Einstiegsprozeduren auf, welche anschließend für die Ethernet-Emulation beschrieben werden.

Die Modul-spezifischen Abschnitte im Objektkode werden durch die folgenden Präprozessormakros definiert. Sie dienen dazu, mit Hilfe des Benutzerprogammes **modinfo** lesbare Textinformationen über das bereits übersetzte Modul zu erfahren.

```
#include <linux/module.h>
MODULE_DESCRIPTION("Emulates an Ethernet network with a shared media.");
MODULE_AUTHOR("Steffen Maier");
MODULE_LICENSE("GPL");
MODULE_PARAM(ethemus, "i");
MODULE_PARAM_DESC(ethemus, "Ethemu: Maximum number of emulation devices.");
```

Einstiegsprozeduren, die der Lademechanismus des Kernels während der Aktivierung bzw. Deaktivierung des Moduls benötigt, werden durch durch die Makros module_init bzw. module_exit kenntlich gemacht.

```
#include <linux/init.h>
static int __init ethemu_init(void)
module_init(ethemu_init);
```

Initialisiert die internen Datenstrukturen inklusive einer durch den Modulparameter *ethemus* bestimmten Anzahl an Emulations-Geräten. Außerdem registriert sich das Modul für den Empfang von Rahmen, die den exklusiv für Ethemu verwendeten Protokolltyp 0x8842 enthalten. Diese Protokollidentifikation tragen die Broadcast-Emulationsrahmen. Auf diese Weise werden alle derartige Rahmen an das Ethemu-Modul – und zwar nur dorthin – weitergeleitet. Dies ist notwendig für die spätere Umkehrung des Tunnelvorgangs, d.h. dem Abstreifen des äußeren Ethernet-Rahmenkopfes. Die Registrierung erfolgt mit Hilfe der in Abb. 6.3 dargestellten Standardschnittstelle dev_add_pack() im Linux-Kernel.

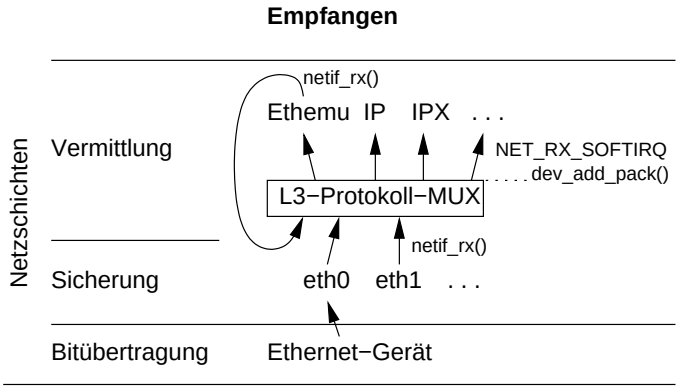


Abbildung 6.3: Einhängen in Empfangsdatenfluß

Zur weiteren Erläuterungen der Abbildung ist ein kleiner Exkurs über die Behandlung empfangener Rahmen im Linux-Kernel notwendig. Nachdem ein Netzadapter einen Rahmen vollständig empfangen hat, löst er eine Unterbrechung aus. Die Unterbrechungsbehandlungsroutine im Gerätetreiber ist dafür zuständig, den Rahmen vom Adapter in den Hauptspeicher zu transferieren. Um nicht unnötig lange die Behandlung weiterer evtl. folgender Unterbrechungen zu blockieren, wird der Rahmen mit Hilfe der Kernel-Funktion netif_rx() in einer Empfangswarteschlange abgelegt und somit die Unterbrechungsbehandlung möglichst schnell terminiert. Die weitere Verarbeitung der Rahmen aus der Warteschlange erfolgt in einem sogenannten SoftIRQ. Dies ist ein in Software implementierter Unterbrechungsmechanismus im Linux-Kernel. Bei bestimmten Auslöseereignissen, jedoch zu vorab unbestimmten Zeitpunk-

ten, wird ein solcher SoftIRQ ausgelöst und behandelt. Die Rahmenempfangswarteschlange wird vom NET_RX_SOFTIRQ weiterverarbeitet, wenn derzeit keine wichtigeren Aufgaben im Kernel anstehen. Bei der Verarbeitung werden die Rahmen anhand ihres Protokolltyps an die entsprechenden Vermittlungsschichtimplementierungen weitergeleitet, was durch „L3-Protokoll-Multiplexer“ in der Abbildung angedeutet wird.

static int __exit ethemu_exit(**void**)
module_exit(ethemu_exit);

Entfernt die angelegten Emulations-Geräte, macht sämtliche Registrierungen aller Art rückgängig und gibt schließlich dynamisch vom Modul belegten Kernel-Speicher frei.

6.2.3 Konfiguration

Die Konfiguration der Ethernet-Emulation erfolgt, wie bereits in der Spezifikation in Abschnitt 6.1.3 beschrieben, durch die IOCTL-Schnittstelle. Behandelt werden die Konfigurationswünsche über Systemrufe aus Benutzerprozessen durch die folgende Funktion im Kernel-Modul:

int ethemu_dev_ioctl(**struct** net_device ∗dev, **struct** ifreq ∗ifr , **int** cmd)

Wickelt alle in der Spezifikation festgelegten Konfigurationskommandos ab. Werteverändernde Kommandos, d.h. mit der Teilzeichenkette SET im Namen, sind dabei nur für Benutzer mit Netzadministrator-Rechten (z.B. root) erlaubt. Alle anderen, lesenden Kommandos sind für alle Benutzer erlaubt. Mit Ausnahme von ETHEMU_SET_DEV greifen praktisch alle Kommandos lesend (GET) oder schreibend (SET/ADD/DEL) auf die in Abschnitt 4.3.1 vorgestellten lokalen Variablen des Emulationsgerätes „dev“ zu. Die erwähnte Ausnahme verändert nicht nur Variablenwerte, sondern nimmt aktiv eine Bindung an ein anderes Netzwerkgerät vor. Diese Aufgabe wird an die folgende Funktion abgegeben:

static void ethemu_bind(**struct** net_device ∗emudev, **struct** net_device ∗realdev)

Versucht, das Emulationsgerät *emudev* an das andere Netzwerkgerät *realdev* zu binden. Hierzu wurde zuvor mit Hilfe des Gerätenamens aus der Konfiguration in struct ethemu_conf ein Zeiger auf *realdev* vom Kernel besorgt. Dieser Zeiger wird in einer lokalen Variable von *emudev* für die spätere Verwendung bei Sendevorgängen abgelegt.

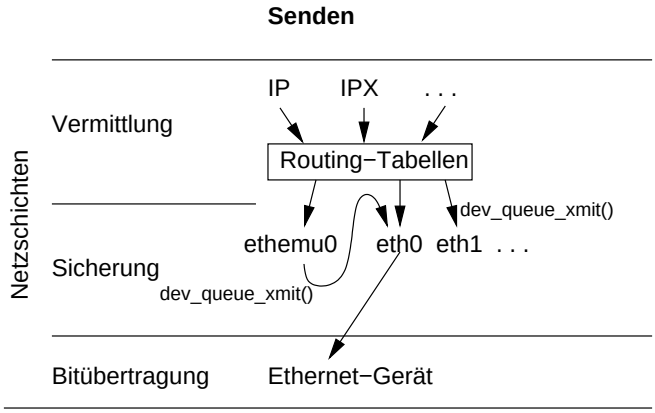


Abbildung 6.4: Einhängen in Sendedatenfluß

Abb. 6.4 zeigt den Datenfluß zu sendender Rahmen. Rahmen aus der Vermittlungsschicht werden bereits inklusive vollständigem Rahmenkopf durch Einträge in Leitwegtabellen an ein Netzwerkgerät

abgegeben. Hier im Beispiel wäre eine Abgabe an das Emulationsgerät ethemu0 möglich. Nach entsprechender Behandlung wird der Rahmen – sofern nicht verworfen – zu gegebener Zeit mit Hilfe der Kernel-Funktion dev_queue_xmit() in die Sendewarteschlange des Gerätes gestellt, auf das zuvor beim Binden ein Zeiger gemerkt wurde. Im Beispiel ist dies das Ethernet-Gerät eth0. Das Einreihen zu sendender Rahmen aus der Vermittlungsschicht in die Sendewarteschlange und deren weitere Behandlung funktioniert analog zum Empfangsfall, wie in Abschnitt 6.2.2 bereits beschreiben, über SoftIRQs.

6.2.4 Algorithmus

Als Ergänzung zu Abb. 6.1 ist in Abb. 6.5 die interne Struktur des Ethemu-Algorithmus graphisch dargestellt. Auch hier sind wiederum Daten- sowie Kontrollfluß und dessen Seiteneffekte durch verschiedenartige Pfeile zu sehen. Zusätzlich geht nun aber der Kontrollfluß mehr ins Detail und zeigt bis auf einige wenige kleine Hilfsfunktionen alle wichtigen Funktionen innerhalb von Ethemu. Diese internen Funktionen beginnen grundsätzlich mit dem Präfix „ethemu_“, um durch einen eigenen Namensraum mögliche Kollisionen mit Symbolen im Kernel bzw. anderen Modulen von vornherein zu vermeiden. Darüber hinaus sind die zwei Schnittstellenfunktionen zum Kernel für das letztendliche Senden von Rahmen durch dev_queue_xmit() bzw. für das Ausliefern von Paketen durch netif_rx() enthalten. Die Angabe der Funktionsnamen dient der Orientierung im Quelltext. Im Vergleich zu Abb. 6.1 erschließen sich die Bedeutungen der einzelnen Funktionen. Für detailliertere Angaben wird an dieser Stelle auf die bereits erwähnte Programmierschnittstellendokumentation und den Ethemu-Quelltext verwiesen.

6.3 Realisierung

Bei der Gesamtumsetzung von Ethemu waren die Bücher [WPR+02, RC01, BBD+01] sowie eine Reihe von Artikeln [Ins02, Ins01, Sal99, kl99, Cox96] und weitere Dokumente aus dem Linux Documentation Project [Aiv01, Pom99, Rus99, Hen02] äußerst hilfreich und stellten praktisch die Dokumentation von Linux-Kernel-Schnittstellen dar. Darüber hinaus ist es unerläßlich, für wichtige Details die Kernel-Quelltexte [T+02] ständig im Blick zu behalten. Hierbei unterstützt LXR (Linux Cross Reference)[1], das die Kernel- aber auch beliebige andere Quelltexte dynamisch in das Hypertextformat HTML umsetzt und so das interaktive Verfolgen von Daten- und Kontrollflüssen sowie effizientes Suchen nach Bezeichnern erlaubt.

Da Programm-Code für die Kernel-Umgebung erstellt wurde, war es notwendig, zusätzlich zum Entwicklungsrechner einen Testrechner zum Ausführen noch ungetesteter Ethemu-Modul-Versionen zur Verfügung zu haben. Eine der besten Vorgehensweisen wäre, wie in [Riz97] beschreiben, sicherlich die Verwendung einer sogenannten diskless Workstation, d.h. eines Rechners ohne eigene Massenspeicher, der über das Netz hochfährt. Dies verhindert zuverlässig die Zerstörung von Dateisystemen bei Kernel-Abstürzen. Das Aufsetzen einer dafür notwendigen Netz-Boot Umgebung ist jedoch sehr aufwendig. Außerdem bestand in der zur Verfügung stehenden Testumgebung im Gegensatz zum NET-Cluster nur eine Netzverbindung pro Knoten, wodurch Emulations- und anderer Netzverkehr sich beeinflussen können. Deshalb kamen als Testknoten normale PCs mit lokaler Festplatte für Betriebssystem, Auslagerungsbereich und Protokolldateien zum Einsatz. Um nach möglichen Kernel-Abstürzen langwierige Dateisystemreparaturen zu verhindern, wurde ein Dateisystem mit Journal-Funktion eingesetzt. Dadurch kann problemlos die Integrität inkorrekt abgehängter Dateisysteme durch schnelles Nachfahren eines Protokolls wiederhergestellt werden.

[1] `http://lxr.linux.no/` bzw. `http://lxr.sourceforge.net/`

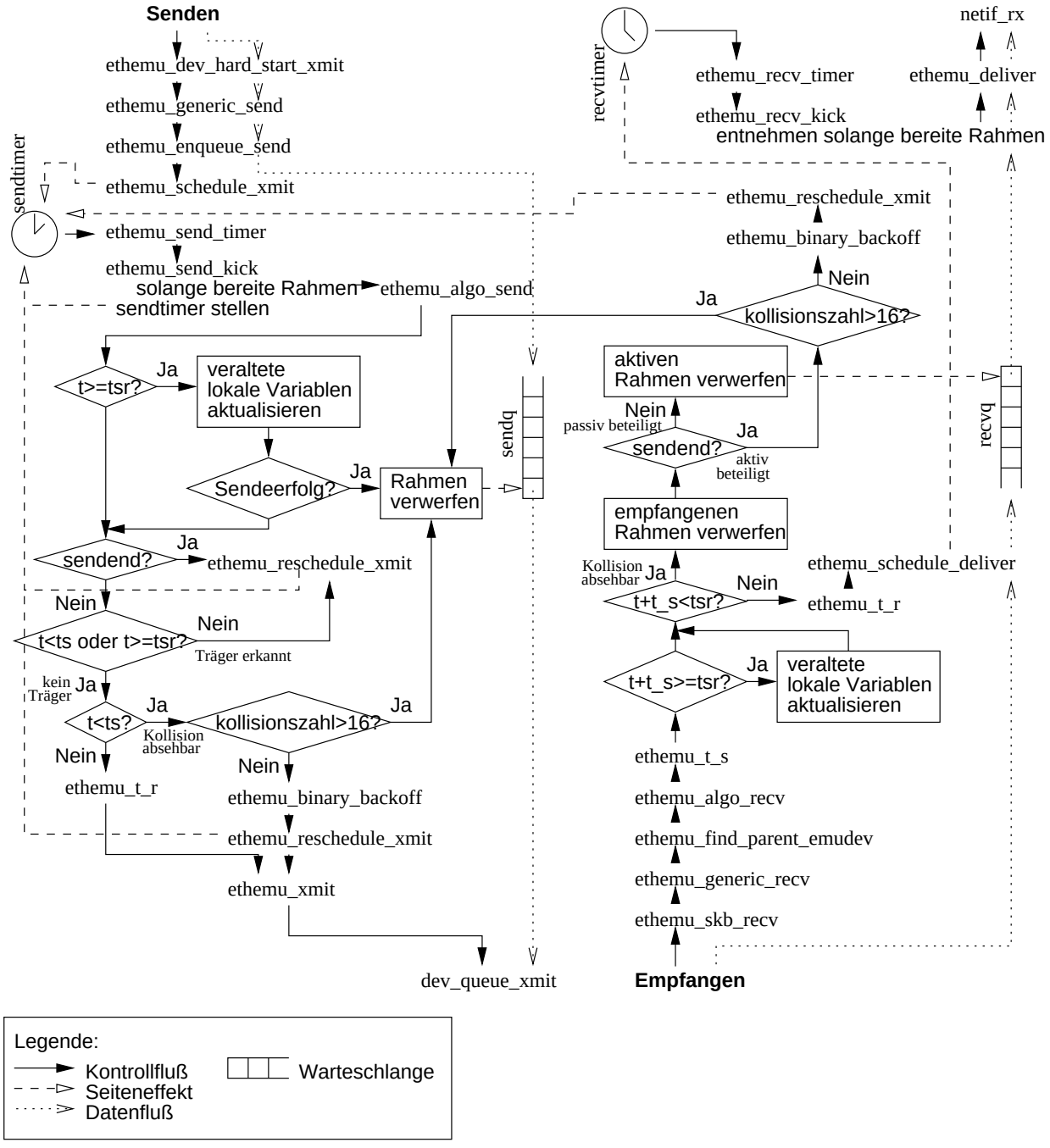


Abbildung 6.5: Interne Struktur des Ethernet-Emulations-Algorithmus

6.4 Fehlersuche und -beseitigung

Zur Fehlersuche wurden Testläufe mit verschiedenen im normalen Betrieb vorkommenden Netzverkehrsmustern durchgeführt, wie beispielsweise ICMP-Echo-Response per **ping**-Befehl oder einfache TCP-Verbindungen per **telnet** auf den Uhrzeitdienst daytime. Während der Testläufe war eine Vielzahl von Debug-Meldungen im Ethemu-Modul aktiviert, die via Systemprotokolldienst auf der jeweils lokalen Festplatte abgelegt wurden. Damit die im Protokoll enthaltenen Zeitstempel eine vergleichende Analyse über die zwei kommunizierenden Testrechner erlauben, wurde mit Hilfe von NTP (Network Time Protocol) die Synchronisation der lokalen Rechneruhren sichergestellt. Da die Zeitstempel eine relativ grobe Sekundenauflösung besitzen, erfolgten die Testläufe mit großen Parametern für Signalausbreitungs-, Contentionslot- und Jambit-Zeit sowie kleinem Parameter für die Übertragungsrate, so daß praktisch ein Emulationsablauf in Zeitlupe erzwungen wurde.

Für den Fall, daß ein Kernelabsturz gefolgt von einer sogenannten „Oops“-Meldung auftrat, war ein Testrechner per serieller Konsole mit dem Entwicklungsrechner verbunden [Smo00]. Auf diese Weise konnten sämtliche Kernel- inklusive Ethemu-Debug- und evtl. auftretenden Oops-Meldungen zuverlässig mitprotokolliert werden, und zwar auch dann, wenn der Systemprotokolldienst die Meldungen nicht mehr rechtzeitig auf die lokale Festplatte durchschreiben konnte. Dies ist hilfreich, um Oops-Meldungen im genauen Wortlaut zur Fehleranalyse an das Werkzeug **ksymoops**[2] von Keith Owens zu übergeben [TW+01]. Das Werkzeug versieht den im Oops enthaltenen numerischen Stacktrace, d.h. die in der Vergangenheit aufgerufenen Funktionen, und den Inhalt des Befehlszeigerregisters, das auf die Fehlerstelle verweist, mit symbolischen Bezeichnern. Zusätzlich wird ein kurzer Maschinen-Code-Ausschnitt ab dem Befehlszeiger disassembliert ausgegeben. Dadurch ist es möglich, die Fehlerstelle im Quelltext zu lokalisieren und zu beheben.

6.5 Testmessungen

Zwar wurde selbstverständlich bereits während der Fehlersuche und -beseitigung (siehe Abschnitt 6.4) dafür gesorgt, daß das Ethernet-Emulationsverfahren möglichst fehlerfrei arbeitet. Dies erlaubt aber noch keine Aussagen über die Realitätsnähe des Emulationsverfahrens selbst. Ausgehend von den Feststellungen in Abschnitt 2.2.1 und Abschnitt 4.2.2 wurden deshalb Messungen bezüglich des Verhaltens des Durchsatzes bei spezifizierten Netzlasten jeweils für verschiedene, emulierte Übertragungsraten vorgenommen. Der folgende Abschnitt beschreibt zunächst die Umgebung, in der die Messungen durchgeführt wurden. Anschließend werden die erhaltenen Ergebnisse ausgewertet.

6.5.1 Meßumgebung

Hardware und Netzkonfiguration

Die Hardware des Meßumgebung bestand aus drei Knoten-PCs vom Typ Intel Pentium-III. Davon waren zwei Knoten mit 700 MHz getaktet und mit einem Fast-Ethernet-Anschluß ausgestattet. Ein Knoten war mit 450 MHz getaktet und verfügte nur über einen 10 MBit/s Ethernet-Anschluß. Via Twisted-Pair-Kabel wurden alle drei Knoten mit einem 10/100 MBit/s Ethernet-Switch untereinander verbunden. Die Knoten kommunizierten mit ihrer jeweiligen Maximalgeschwindigkeit im Full-Duplex-Modus. Als Linux-Distribution diente ein Debian Potato 2.2r6 mit Zusatzpaketen zur Unterstützung des eingesetzten, neueren Kernel 2.4.18. Alle unbenötigten Dienste waren deaktiviert, so daß die Messungen nicht durch

[2] `ftp://ftp.kernel.org/pub/linux/utils/kernel/ksymoops/v2.4/`

unnötigen, zusätzlichen Netzverkehr verfälscht werden. Ein markiertes VLAN mit der Identifikationsnummer 42 war jeweils an die Ethernet-Geräte gebunden. Die Emulationsgeräte waren wiederum jeweils an das VLAN-Gerät gebunden, so daß sich eine Situation wie in Abb. 6.2 ergab. Die den Messungen zugrundeliegende Netztopologie ist in Abb. 6.6 zu sehen.

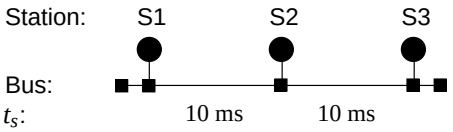


Abbildung 6.6: Netztopologie für die Messungen

Ethemu wurde auf den Knoten jeweils wie folgt konfiguriert:

- „**eeconfig** ethemu0 -d vlan42“ bindet Emulationsgerät an VLAN 42.
- „**eeconfig** ethemu0 -a <MAC-Adresse>,20000000“ zur Konfiguration der auf der Topologie basierenden Signalausbreitungszeiten für die weit entfernten Stationen wie zwischen S1 und S3 bzw. „**eeconfig** ethemu0 -a <MAC-Adresse>,10000000“ für direkt benachbarte Stationspaare. Bei dem Wert 10000000 ns handelt es sich um das kleinste exakt meßbare Zeitintervall von 10 ms innerhalb der Ethemu-Implementierung, weshalb dieser Wert als minimale Signalausbreitungszeit zwischen zwei Stationen für die Messungen herangezogen wurde.
- „**eeconfig** ethemu0 -b <Übertragungsrate>“ zur Konfiguration der Übertragungsrate je nach Meßreihe.
- „**eeconfig** ethemu0 -c 40000000“ stellt eine Contentionslot-Dauer von 40 Millisekunden ein, was exakt der doppelten Ausbreitungszeit eines Signals vom einen Ende der Topologie zum anderen entspricht. Dies ist die Minimaldauer bzw. besitzt umgekehrt die Topologie die längste Ausdehnung bei der Kollisionen noch rechtzeitig erkannt werden können.
- „**eeconfig** ethemu0 -j 40000000“ stellt eine Jambit-Dauer von 40 Millisekunden ein und wurde analog zur Contentionslot-Dauer gewählt. Bei realem Ethernet ist dieser Wert zwar ein Vielfaches eines Contentionslots, jedoch wurde darauf verzichtet, da die absoluten Zeitdauern bereits weit über den beispielsweise bei 10 MBit/s Ethernet eingesetzten liegen.
- „**eeconfig** ethemu0 -m 64“ konfiguriert die minimale Rahmengröße auf 64 Bytes, wobei diese Funktionalität in Ethemu noch nicht implementiert ist und somit keine weiteren Auswirkungen hat.
- „**ifconfig** ethemu0 txqueuelen 0“ kürzt die Kernel-interne Warteschlange für zu sendende Rahmen auf eine Länge von Null und wird somit praktisch deaktiviert, um v.a. für langsame Übertragungsraten Einflüsse durch Zwischenpufferung zu verhindern.

Lastgenerator-Software

In Ermangelung einer geeigneten Software zur Generierung spezifizierter Netzlasten auf Sicherungsschicht wurde ein eigener Lastgenerator entwickelt. Der Generator unterstützt die Spezifikation folgender Parameter:

- Name des Netzwerkgerätes, auf dem Last generiert werden soll.

- Nominale Übertragungsrate des Netzes, d.h. die Übertragungsrate, die wie oben beschrieben für das Emulationsgerät jeweils eingestellt wurde.
- Zu generierende Gesamtlast über alle beteiligten Stationen, die gleichwertig anteilig auf jede einzelne Station verteilt wird.
- Anzahl der beteiligten Stationen.
- Minimale und maximale Dauer von Pausen zwischen Sende-Bursts in Nanosekunden.
- Minimale und maximale Rahmengröße in Bytes.
- Eine beliebige Anzahl von MAC-Adressen als Zielstationen, an die Last in Form von Unicast-Rahmen gesendet wird.

Die spezifizierte Last erzeugt der Generator, indem er eine interne „Last-Uhrzeit“ t_{last} mitführt. Diese Zeit wird beim Start auf die aktuelle Systemzeit t_{start} gestellt. Nach dem Start schreitet t_{last} um die Rahmensendedauern $t_r(f)$ der jeweiligen Rahmen f, summiert über alle bisher gesendeten Rahmen F, voran. Die Rahmensendedauer $t_r(f)$ berechnet sich aus effektiver Übertragungsrate b_{eff} und Länge r (vgl. Gleichung 4.2). Dabei setzt sich die effektive Übertragungsrate b_{eff} zusammen aus nominaler Übertragungsrate b, Gesamtlast l und Anzahl der Stationen n: $b_{\text{eff}} = b \cdot l/n$.

$$t_{\text{last}} = t_{\text{start}} + \sum_{f \in F} t_r(f) = t_{\text{start}} + \sum_{f \in F} \frac{r(f)}{b_{\text{eff}}} = t_{\text{start}} + \sum_{f \in F} \frac{r(f)}{b \cdot l/n} \tag{6.1}$$

Solange t_{last} kleiner oder gleich der aktuellen Rechneruhr ist, hat der Generator seine Sollast noch nicht erzeugt und generiert deshalb solange Rahmen, bis die Zeitbedingung nicht mehr erfüllt ist. Die Rahmen haben eine Länge, die gleichverteilt zufällig aus dem Intervall zwischen minimaler und maximaler Rahmenlänge (jeweils inklusive) gewählt wird. Nachdem t_{last} durch genügend gesendete Rahmen zur Realzeit aufgeholt hat, schläft der Generator für eine Zeitdauer, die gleichverteilt zufällig zwischen minimaler und maximaler Pausendauer (jeweils inklusive) liegt. Anschließend beginnt der Vorgang wieder von vorn.

Auf diese Weise erzeugt der Lastgenerator burst-artigen Netzverkehr mit unterschiedlichen Rahmengrößen, der über die Zeit gemittelt der spezifizierten Last entspricht.

Zur Realisierung des Programmes in der Programmiersprache C wurde die Programmierbibliothek Libnet[3] von Mike D. Schiffman verwendet. Sie erlaubt unter UNIX-Betriebssystemen unabhängig von den spezifischen Eigenheiten der jeweiligen System-Programmierschnittstelle u.a. das Senden von Rahmen auf Sicherungsschicht durch direkte Übergabe an Netzwerkgerätetreiber.

Protokollier- und Meß-Software

Ein Shell-Skript, das über eine Menge von Übertragungsraten und für jede davon wiederum über eine Menge von Netzlasten iteriert, wurde zur Automatisierung der Meßläufe eingesetzt. In jeder Iteration werden so über das Netz via SSH (Secure SHell) die Knoten konfiguriert und jeweils ein Lastgenerator gestartet. Möglichst gleichzeitig wird auf dem mit 10 MBit/s angebundenen Knoten ein Netz-Sniffer zur Protokollierung der erfolgreich auf dem Ethemu-„Medium“ übertragenen Rahmen ausgeführt. Nachdem der Sniffer eine repräsentative Anzahl an Rahmen mitgelesen hat, werden er und die Lastgeneratoren wieder gestoppt und die nächste Iteration kann nach kurzer Wartezeit beginnen.

[3] `http://www.packetfactory.net/libnet`

Die Automatisierung mit Hilfe des Skriptes dient dem unbeaufsichtigten Durchführen von Meßreihenaufzeichnungen. Wie auch in [KGM+01] angemerkt, können Messungen unter einer Emulation sehr zeitaufwendig sein. Dabei wird die Zeitdauer durch die Geschwindigkeit des emulierten Netzes bestimmt. Im Falle der Messungen von Ethemu konnte allein das Erhalten eines einzigen Meßwertes einer Reihe u.U. mehrere Stunden benötigen. In einem solchen Fall ist es umso notwendiger, daß durch unbeaufsichtigten Ablauf der die Messungen Durchführende die Zeit sinnvoll anderweitig nutzen kann und nicht durch Beaufsichtigung aufgehalten wird.

Als Netz-Sniffer zur Erzeugung von „Traces“ des Netzverkehrs wurde **tcpdump**[4] verwendet. Dieser wurde ursprünglich von der Network Research Group (NRG) der Information and Computing Sciences Division (ICSD) am Lawrence Berkeley National Laboratory (LBNL) entwickelt. Zur späteren Auswertung der Sniffer-Protokolle kam **tcpstat**[5] von Paul Herman zum Einsatz. Es ermöglicht u.a. das Einlesen von pcap-kompatiblen Sniffer-Protokollen wie sie tcpdump erzeugt zur Erzeugung von Statistiken. Die somit ermittelten Übertragungsraten zu jeder Meßiteration entsprechen dem verfügbaren Durchsatz in der Emulation.

Meßreihen wurden für die in Tab. 6.1 angegebenen Übertragungsraten, Anzahlen an protokollierten, repräsentativen Rahmen und Netzlasten erstellt. Zusätzlich dient eine Meßreihe eines ungeswitchten 10BaseT-Ethernets aus der ansonsten identischen Meßumgebung als Vergleichskriterium. Die Auswahl der Übertragungsraten erfolgte nach dem Gesichtspunkt der Zeitgranularität der Ethemu-Implementierung. Da die Rechneruhr sowie Kernel-Timer unter Linux standardmäßig eine Auflösung von 10 Millisekunden besitzen, stellt 100 Bit/s prinzipiell die höchste Rate dar, bei der Kollisionen Bit-genau erkannt werden können (vgl. hierzu Tab. 5.1). Diese äußerst niedrige und deshalb eigentlich weniger interessante Rate wurde als kleinste in die Meßreihen mitaufgenommen, da davon ausgegangen werden kann, daß durch die Bit-genaue Zeitmessung die Emulation theoretisch das realitätsnächste Verhalten aufweisen müßte. Höhere Raten werden in Ethemu intern in Byte/s behandelt, wodurch sich die jeweils durch acht teilbaren, weiteren Übertragungsraten ab 800 Bit/s aufwärts ergeben. Bei 800 kBit/s ergaben sich Stabilitätsprobleme in der Emulation, so daß weitere Meßreihen leider nicht möglich waren.

Übertragungsrate	**Rahmenanzahl**	**Netzlasten**										
		Ethemu										
100 Bit/s	50	0,1	0,5	1	2	5	10	20	100			
800 Bit/s	100	0,1	0,5	1	2	5	10	15	20	50	100	1000
8 kBit/s	100	0,1	0,5	1	2	5	10	15	20	50	100	1000
80 kBit/s	100	0,1	0,5	1	2	5	10	15	20	50	100	1000
800 kBit/s	1000	0,1	0,5									
		10BaseT-Ethernet										
10 MBit/s	100000	0,1	0,5	1	2	5	10	15	20	50	100	1000

Tabelle 6.1: Für die Meßreihen verwendete Übertragungsraten und Netzlasten

Über alle Meßreihen wurde der Lastgenerator jeweils mit den in Tab. 6.2 angegebenen konstanten Parametern betrieben.

6.5.2 Meßergebnisse

In Abb. 6.7 sind die Meßwertreihen für den gemessenen Durchsatz über der Netzlast aufgetragen.

[4] http://www.tcpdump.org/

[5] http://www.frenchfries.net/paul/tcpstat/

Parameter	Wert	Einheit
Anzahl der beteiligten Stationen	3	–
minimale Dauer von Pausen	0	ns
maximale Dauer von Pausen	10000000	ns
minimale Rahmengröße	64	Byte
maximale Rahmengröße	1500	Byte

Tabelle 6.2: Konstante Betriebsparameter des Lastgenerators für die Meßreihen

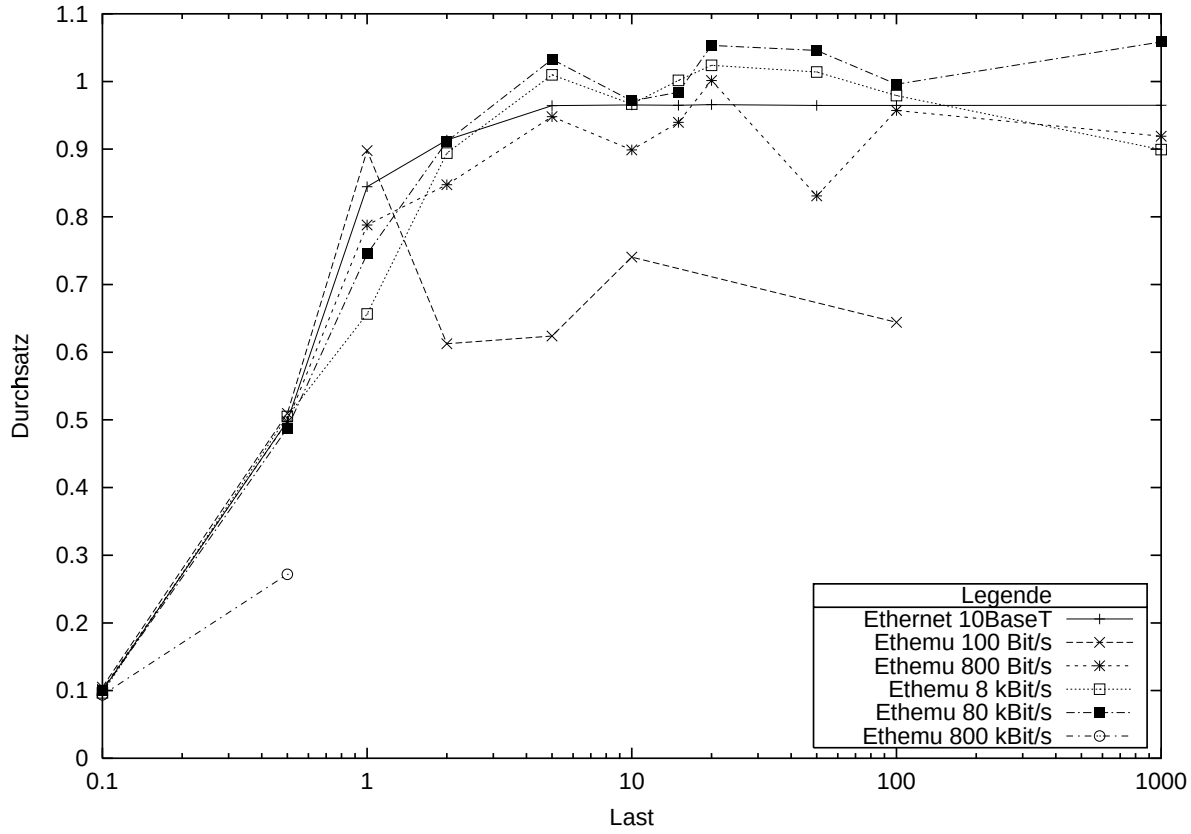


Abbildung 6.7: Vergleich des Lastverhaltens bei verschiedenen Übertragungsraten

Grundsätzlich sind die Meßkurven unter Vorbehalt zu betrachten. Im Vergleich zu Abb. 4.4 fällt zunächst auf, daß in den Meßkurven bei Lasten höher als ca. 15 kein Abfall des verfügbaren Durchsatzes auftritt. Hier spielen sicherlich zwei Aspekte eine Rolle.

Zum einen basieren die Kurven in Abschnitt 4.2.2 auf theoretischen, mathematischen Analysen. Als Grundlage wird eine Poisson-Verteilung der ankommenden Rahmen, die zudem alle stets dieselbe Größe aufweisen, angenommen. Daß realer Rechnernetzverkehr selten Poisson verteilt, sondern selbstähnlich ist, zeigen [PF94, WTSW95].

Außerdem gehen die Analysen von unbegrenzten Warteschlangen aus, was bei einer realen Implementierung und somit auch Ethemu unmöglich ist. Denn sobald mehr Netzlast erzeugt wird, als auf dem Medium fehlerfrei gesendet werden kann, müssen die wartenden Rahmen in eine Warteschlange eingereiht werden. Allein durch den endlichen Speicher ist eine Schlange einer realen Implementierung in ihrer Größe begrenzt. Rahmen, die der Kernel von dem zu Meßzwecken eingesetzten Lastgenerator für ein Ethemu-Gerät erhält, müssen verworfen werden, sobald die Warteschlange voll ist und der Generator nicht blockieren soll.

Zum anderen konnte bedingt durch die zur Verfügung stehende Meßumgebung stets von nur drei und nicht mehr Knoten Last auf das Netz gegeben werden. Somit war die Erzeugung höherer Last ausschließlich durch Erhöhen der Rahmen-„Ankunftsrate“ möglich, jedoch nicht durch Erhöhen der Knotenanzahl. Dabei wäre bei einer größeren Knotenanzahl mit einer höheren Wahrscheinlichkeit für Kollisionen zu rechnen, da der Fall des quasi gleichzeitigen Sendens von Stationen wahrscheinlicher wird.

Das Verhalten bei einer emulierten Übertragungsrate von 100 Bit/s weist entgegen den Erwartungen, daß es hier am realitätsnächsten sein müßte, deutliche Abweichungen vom realen Ethernet auf. Es ist schwer zu interpretieren, woher dieses Fehlverhalten rührt. Ein Blick in die Meßprotokolle zeigt ab einer Last von 2 eine überdurchschnittlich hohe Zahl an Kollisionen. Der Effekt auf den Durchsatz ist durch den Einbruch der Kurve auszumachen. Jedoch bleibt unklar, warum die Kollisionsanzahl plötzlich stark erhöht ist.

Die Reihe für eine Übertragungsrate von 800 kBit/s kann durch die dort auftretende Instabilität der Emulation sowie mangels weiterer Meßwerte nicht zur Beurteilung in Bezug auf die maximal mögliche Rate, bei der sich die Emulation noch sinnvoll verhält, herangezogen werden.

Bei Übertragungsraten zwischen 800 Bit/s und 80 kBit/s ist eine deutliche Ähnlichkeit zur Vergleichskurve des realen Ethernet zu erkennen. Offenbar läßt sich Ethemu mindestens mit Raten, die um beinahe bis zu knapp drei Größenordnungen über 100 Bit/s liegen, weitgehend realitätsnah in Bezug auf das Durchsatz-Lastverhalten betreiben.

Die bei höheren Lasten vorhandenen Abweichungen von der Ethernet-Vergleichskurve sind zurückzuführen auf die relativ grobe Timer-Granularität der Ethemu-Implementierung. Die Auslösegenauigkeit der verwendeten Kernel-Timer beträgt nur 10 ms. Dadurch sind verspätete Timer-Ereignisse möglich. Dies führt zu vergrößerten Zeitabständen zwischen gesendeten Rahmen und ist somit eine möglich Ursache für Durchsatzwerte, die geringer als bei realem Ethernet sind.

Außerdem wurde auf Details aus dem IEEE-Standard 802.3 wie Interframe-Gap, d.h. einer kurzen Sendepause bei direkt hintereinander ausgesandten Rahmen, verzichtet. Diese Pause ist in einer Größenordnung – beispielsweise 9,6 μs bei Ethernet mit 10 MBit/s –, die weit unter der Timer-Granularität liegt. Folglich hätte ein Abwarten eines solch kurzen Zeitintervalls mit Hilfe der Timer keinen Sinn, da im schlimmsten Fall eine ganze Timer-Periode von 10 ms bis zur Auslösung vergehen könnte. Die andere Alternative wäre eine sofortige Auslösung, was jedoch dem Verzicht auf eine solche Pause entspricht. Der Verzicht kann durch unmittelbar direkt hintereinander ausgesandte Rahmen zu Durchsatzwerten führen, die größer als bei realem Ethernet sind.

Zu den Ungenauigkeiten innerhalb von Ethemu kommen Meßfehler hinzu. Außerdem hängt die Ver-

teilung der Rahmengrößen im Lastgenerator vom dort eingesetzten Zufallsgenerator ab. Da diese meist nur Pseudozufallszahlenfolgen erzeugen, ist es möglich, daß während der Meßzeit die Rahmengrößen nicht exakt gleichverteilt über das angegebene Intervall sind. In der Folge führt dies bei einer Durchschnittsrahmengröße unter dem Sollmittelwert zu einer geringeren generierten Last, was sich natürlich auf einen niedrigeren gemessenen Durchsatz auswirkt. Umgekehrt gilt für Durchschnittsrahmengrößen über dem Sollmittelwert, daß eine zu hohe Last generiert wird, die sich durch höheren gemessenen Durchsatz auswirkt.

Eine bestimmte Last kann vom Generator nur durch die Zeitabstände zwischen gesendeten Rahmen generiert werden, da stets nur ganze Rahmen und keine einzelnene Bytes gesendet werden können. Vor allem bei niedrigen emulierten Übertragungsraten ergibt sich eine relativ lange Rahmenübertragungszeit und somit größere Zeitabstände zwischen gesendeten Lastrahmen. Das Meßergebnis ist aber der Durchsatz in Bit/s, d.h. er ergibt sich aus den im Mittel gesendeten Bytes aller Rahmen während der Meßdauer. Nach dem Gesetz der großen Zahlen wird der gemessene Wert genauer, je mehr Rahmen in die Messung einfließen, da sich dann die Wahrscheinlichkeit der vorkommenden – eigentlich gleichverteilten – Rahmengrößen stabilisiert. Da jedoch gerade bei niedrigen emulierten Raten die Meßdauer, selbst über wenige korrekt übertragene Rahmen, sehr lang sein kann, wurde auf noch längere Meßzeiträume verzichtet.

Die Tatsache, daß der gemessene Durchsatz bei emulierten 8 bzw. 80 kBit/s stellenweise größer eins liegt, sollte nicht allzu negativ bewertet werden. Es bedeutet nichts anderes, als daß die gemessene erzielte Übertragungsrate teilweise geringfügig über der eingestellten emulierten Rate liegt.

Der Ansatz ist also durchaus für die Emulation von Ethernet geeignet, wenngleich er sicherlich in der Implementierung in Bezug auf feinere Zeitgranularität noch verbessert werden kann. Bis dahin ist Ethemu für emulierte Übertragungsraten im Bereich von ungefähr 800 Bit/s bis hinauf zu 80 kBit/s unmittelbar verwendbar.

Kapitel 7

Ausblick

Zunächst werden in diesem Kapitel mögliche Erweiterungen und Verbesserungen für Ethemu aufgezeigt und zukünftige Arbeiten identifiziert. Zum Abschluß dieser Ausarbeitung wird in der Schlußfolgerung der Inhalt der Arbeit rekapituliert und anhand der Ergebnisse das Fazit gezogen.

7.1 Erweiterungen und Verbesserungen

Eine Reihe möglicher Erweiterungen und Verbesserungen für Ethemu v.a. in Bezug auf die höchste, sinnvoll emulierbare Übertragungsrate werden in den folgenden Abschnitten besprochen.

7.1.1 Feinere Zeitgranularität

Eine feinere Zeitgranularität dient dem sicheren Erkennen von zeitlichen Rahmenüberschneidungen anhand derer in der Emulation Kollisionen erkannt werden. Im Idealfall weist die Granularität der in Ethemu verwendeten Zeit höchstens eine Bit-Dauer auf, so daß bereits die Überschneidung eines einzigen Bits erkannt werden kann. Verbesserungen mit diesem Ziel lassen sich auf unterschiedliche Art im Datenpfad empfangener Rahmen sowie im Datenpfad zu sendender Rahmen vornehmen.

Feinere Zeitgranularität im Eingangsdatenpfad

Auf zweierlei Wegen kann für die Gültigkeit der Annahme, daß die Übertragungszeit von Rahmen im Emulationsnetz vernachlässigbar ist, gesorgt werden. Zum einen durch eine hohe physikalische Übertragungsrate in der Emulationsumgebung, so daß die Rahmensendedauer gering ist. Zum anderen muß die Latenzzeit bis zur Verarbeitung eines empfangenen Rahmens so gering wie möglich gehalten werden, denn diese Zeit addiert sich zur Rahmensendedauer.

Die derzeitige Ethemu-Implementierung setzt, wie in Abschnitt 6.2.2 beschrieben, vorhandene Standardschnittstellen des Linux-Kernel ein, um empfangene Rahmen abzufangen. Allerdings kann im schlechtesten Fall zwischen der Unterbrechung des Netzadapters und der Ausführung des NET_RX_SOFTIRQ bis zu einer vollen Timer-Periode liegen. Standardmäßig sind dies 10 ms. Tatsächlich wurde während den Messungen beobachtet, daß diese Verzögerung stets unter 30 μs lag und im Mittel ungefähr 20 μs beträgt, wobei die dazu verwendete Zeitmessung eine Mikrosekunde auflösen kann. Da auch eine solche Verzögerung immer noch unerwünscht ist, bietet die Umgehung des SoftIRQ-Mechanismus einen Ausweg. Hierzu kann der Kernel-Quelltext so modifiziert werden, daß eine zentrale Hilfsfunktion, die von allen Netzwerkgerätetreibern innerhalb der Unterbrechungsbehandlung für empfangene Rahmen aufgerufen wird, auf eine eigene Implementierung dieser Funktion umgeleitet werden kann. Eine solche

Funktion ist netif_rx(). Sie wird von allen Netzwerkgerätetreibern verwendet, um empfangene Rahmen in den Kernel einzuspeisen. Mit einer solchen Modifikation ist das Abfangen von Rahmen praktisch ohne weitere vermeidbare Verzögerungen möglich.

Um diese Zeitersparnis gewinnbringend zu nutzen, wird außerdem statt den häufig im Kernel verwendeten „jiffies“, welche die Systemzeit nur nach Timer-Perioden auflösen, der Inhalt des TSC-Registers (Time Stamp Counter) moderner Prozessoren verwendet. Die Prozessorfamilie des Intel Pentium stellt ein solches Register zu Verfügung. Es handelt sich dabei um einen Zähler, der in der Geschwindigkeit der Prozessortaktzyklen inkrementiert wird.

Nun kann also ein eben empfangener Rahmen umgehend mit einem sehr feingranularen Zeitstempel versehen werden und anschließend sofort in der Emulation das weitere Vorgehen für jenen Rahmen entschieden werden. Allerdings ist nun sehr genau darauf zu achten, daß die Entscheidungsfindung innerhalb der Emulation äußerst schnell terminiert. Alle Berechnungen finden durch die Modifikation nämlich in der Behandlungsroutine einer wichtigen Hardware-Unterbrechung statt. Solange sind weniger wichtige Unterbrechungen gesperrt und könnten verloren gehen oder zu spät behandelt werden.

Feinere Zeitgranularität im Ausgangsdatenpfad

Bei zu sendenden Rahmen geht es vor allem darum, möglichst genau zum vorausberechneten Zeitpunkt einen Rahmen über das Emulationsnetz schicken zu können. Das betrifft vor allem Sendungen, bei denen ein Rahmen direkt nach dem anderen ausgegeben werden soll, sowie die Umsetzung der 1-Persistenz bei Ethernet, d.h. dem unmittelbaren Senden eines wartenden Rahmens nachdem der Träger vom Medium verschwunden ist. Dazu sind genaue Timer zur Abarbeitung der Sendewarteschlange in Ethemu notwendig.

Passend zu den im vorangehenden Abschnitt erwähnten TSC-Zeitstempeln existieren Timer-Implementierungen, die Auslöseunterbrechungen aus dem bei modernen Prozessoren aus der Pentium-Familie vorhandenen APIC (Advanced Programmable Interrupt Controller) beziehen. Die Hardwaretimer des APIC arbeiten ebenfalls wie ein TSC-Register mit Taktzyklengeschwindigkeit. Zwar ist Unterstützung für den APIC in neueren Kernel-Versionen vorhanden, jedoch erfordert die Nutzung seiner Timer Modifikationen am Kernel-Quelltext. Entsprechende Patches gibt es beispielsweise in Form der UKA-APIC-Timer[1] von der Universität Karlsruhe [WPR+02]. Eine ähnliche Funktionalität ist in KURT[2] (Kansas University Real-Time Linux) enthalten, wobei APIC-Timer hier nur einen Teil einer umfassenderen Echtzeitunterstützung für Benutzerprogramme darstellen.

Analog zur Vorgehensweise im Eingangsdatenpfad wird es hier notwendig, den sendeseitigen Soft-IRQ zu umgehen, um den Vorteil genauerer Timer ausnutzen zu können. Hierzu ist vorstellbar, Rahmen direkt an die Sendefunktion hard_start_xmit() des Netzadaptertreibers zu übergeben, statt sie mit Hilfe der Kernel-Funktion dev_queue_xmit() in die Sendewarteschlange zu stellen.

Mögliche Einschränkungen

Sowohl im Eingangs- wie auch im Ausgangsdatenpfad weisen die Umgehungen der jeweiligen Soft-IRQs eine Einschränkung auf. Die in den zwei vorangehenden Abschnitten beschriebene Vorgehensweise funktioniert, solange Ethemu-Gerätetreiber direkt an ein reales Gerät wie beispielsweise einen Ethernet-Treiber gebunden sind. Liegt dazwischen jedoch noch eine weitere Indirektionsstufe, z.B. über

[1] `http://www.telematik.informatik.uni-karlsruhe.de/forschung/apic/apic_timer-index.html` bzw. neuere Versionen unter `http://www.vincent.oberle.com/apic_timer.html`

[2] `http://www.ittc.ku.edu/kurt/downloads-noframes.html`

den VLAN-Gerätetreiber, so verwendet dieser zwangsläufig wiederum die alten SoftIRQ-basierten Methoden zum Abfangen und Senden von Rahmen. Eine Lösung besteht darin, alle zusätzlichen Gerätetreiberschichten, die auf dem untersten, realen Treiber aufsetzen, wie oben beschrieben zu modifizieren. Im Falle des VLAN-Treibers sollte dies relativ problemlos möglich sein. Er existiert in Form eines Kernel-Moduls und kann somit unabhängig vom Kernel verändert und sogar während der Laufzeit ausgetauscht werden.

Vereinfachte Variante

Eine feinere Zeitgranularität in wesentlich geringerem Verbesserungsumfang als oben beschrieben läßt sich einfach durch Verkürzen der Timer-Unterbrechungsperiode im Linux-Kernel erreichen. Hierzu ist es ausreichend, das Präprozessormakro HZ in der Quelltextdatei include/asm-i386/param.h von 100 auf 1000 abzuändern und einen neuen Kernel zu übersetzen. Über die Auswirkungen auf Timer gibt es jedoch unterschiedliche Aussagen in der Literatur. So wird beispielsweise in [RC01] angemerkt, daß trotz häufigerer Timer-Unterbrechungen die Verarbeitung der registrierten Timer-Funktionen nach wie vor nur alle 10 Millisekunden stattfindet. Als Grund dafür wird angegeben, daß sich der Overhead zur Behandlung der Timer-Funktionslisten nicht lohne. Dies würde bedeuten, daß dann mit der Modifikation eigentlich nur die Zeitgranularität der jiffies nicht aber der Kernel-Timer verbessert würde.

Diese Variante hat außerdem einige Nachteile. Durch häufigere Unterbrechungsbehandlungen erhöht sich der Overhead [ADLY95]. Dieser ist jedoch bei heutigen Workstations gering genug [Riz97]. Allerdings zieht die Modifikation weitere Auswirkungen nach sich. Der Prozeß-Scheduler nimmt häufigere Kontextwechsel vor, die teuer in Bezug auf Rechenzeit sind. Außerdem werden andere Kernel-Timer wie beispielsweise TCP-Timer beeinflußt [HR02]. Durch die verbesserte Timer-Genauigkeit kann sich evtl. ein ungewohntes Verhalten von TCP einstellen.

7.1.2 Fehlerausgleich über die Zeit

Wenn Timer-Auslösezeitpunkte ungenauer als die eigentlichen Sollzeitpunkte sind, kommt es über längere Zeit gesehen vor, daß einmal zu früh und ein anderes mal zu spät ausgelöst wird. Durch explizites Verschieben des Sollzeitpunktes läßt sich über die Zeit gemittelt der absolute Fehler minimieren. Da aber gerade für das Senden genaue Timer benötigt werden, würde absichtlich zu frühes Aussenden eines Rahmens von allen anderen Stationen u.U. als Kollision interpretiert werden. Dies würde die Funktionsfähigkeit der Emulation außer Gefecht setzen. Ebenso könnte ein zusätzlich verzögert gesandter Rahmen dazu führen, daß in der Zwischenzeit eine andere Station aufgrund des Fehlens eines Trägers auf dem Medium zu senden beginnt. Auch dieses würde im Nachhinein eine falsch erkannte Kollision zur Konsequenz haben. Schließt man tatsächlich verspätete Sendungen aus, so könnten möglicherweise durch das Transportieren der absichtlichen, relativen Verschiebung des Sollzeitpunktes im Tunnelrahmen, alle Stationen über den Fehlerausgleich unterrichtet werden und trotzdem korrekt in Bezug auf die Emulation reagieren.

Alles in allem scheint dieser Ansatz trotzdem fragwürdig, da ein solches Vorgehen beispielsweise zur Übertragungsratenlimitierung eingesetzt werden soll. Im Falle von Ethemu ergibt sich die Übertragungsratenlimitierung allerdings nur implizit aus der Nachbildung des MAC-Verfahrens. Primär werden Rahmenzeitstempel verglichen. Die Übertragungsrate wirkt sich nur in Form der Rahmensendedauer einzelner Rahmen aus. Insofern sind Zweifel angebracht, ob dieser Ansatz tatsächlich zu einer Verbesserung von Ethemu führen kann.

7.1.3 Rahmenüberschneidungserkennung bei grober Zeitgranularität

Sollte beispielsweise bei hohen emulierten Übertragungsraten die Erkennung von Rahmenüberschneidungen durch eine zu grobe Zeitgranularität tatsächlich inakzeptabel werden, wäre es vorstellbar, die Überlappung anhand eines ε-Intervalls festzustellen. Auch hier tritt wieder ähnlich wie im vorangehenden Abschnitt 7.1.2 das Problem auf, daß alle Stationen konsistente Entscheidungen bezüglich Überlappungen treffen müssen.

7.1.4 Exaktere Sendezeitpunkte durch real gedehnten Träger

Ein sehr interessanter Ansatz für genauere Sendezeitpunkte nach dem Abhorchen des Mediums auf einen Träger stammt von Christoph Ruffner. Die Idee ist, in diesem Fall auf Timer zu verzichten und stattdessen den physikalischen Träger auf dem Emulationsnetz so lange wie den emulierten auszudehnen. Hierzu wird wie bisher ein Rahmen getunnelt gesendet. Auf atomare Weise wird nun aber direkt im Anschluß quasi ein Füllrahmen hinterhergeschickt, dessen Rahmensendezeit zusammen mit dem ersten der Rahmensendezeit in der Emulation entspricht. Dies führt beim Empfänger dazu, daß er nach dem ersten Rahmen, dessen Übertragungszeit vernachlässigt wird, den Trägerbeginn in Form einer Unterbrechung von der Netzkarte feststellen kann. Nach dem Folgerahmen mit nutzlosem Füllinhalt wird wiederum eine Unterbrechung ausgelöst, die nun aber das Ende des Trägers markiert. Da hier offensichtlich schnell auf Unterbrechungen für empfangene Rahmen reagiert werden muß, sollte dieser Ansatz mit den Optimierungen für den Eingangsdatenpfad aus Abschnitt 7.1.1 kombiniert werden.

Ein wohl unlösbares Problem in der geswitchten NET-Umgebung bringt der Folgerahmen mit dem gedehnten Träger mit sich. Durch die unvermeidliche Serialisierung aller Rahmen in den Puffern des Switches, scheint es nicht mehr möglich, eine Kollision zu erkennen. Dies ist in der bisherigen Implementierung möglich, da die Rahmensendezeit vernachlässigt werden kann und somit während des Andauern des emulierten Trägers weitere Rahmen gesendet und empfangen werden können. Dabei ergibt sich in der Emulation die Überlappung, die als Kollision interpretiert wird. Wenn nun aber der Träger real so lange andauert wie emuliert, werden bereits im Switch alle Rahmen, die währenddessen gesendet werden, zwangsläufig in FIFO-Ordnung serialisiert und es kommt zu keiner zeitlichen Überlappung mehr.

7.2 Zukünftige Arbeiten

Die wichtigste Arbeit an Ethemu ist zunächst die Sicherung der Stabilität bei höheren emulierten Übertragungsraten. Es ist anzunehmen, daß die aufgetretenen Instabilitäten auf Wettbewerbsbedingungen unterschiedlicher Ausführungsstränge beim Zugriff auf gemeinsame Variablen zurückzuführen sind. Insofern sollte das Problem durch Sicherstellen des Exklusivzugriffes auf kritische Abschnitte im Programm zu beheben sein. Die dazu notwendigen Sperrungen wären außerdem notwendige, jedoch nicht hinreichende Grundvoraussetzung für die Fähigkeit, Ethemu auf SMP (Symmetrischen Multi-Prozessor) Systemen auszuführen.

Mit diesen Voraussetzungen ist die Erweiterung um die vorgeschlagenen Verbesserungen aus Abschnitt 7.1.1 für feinere Zeitgranularität mit Hilfe von TSC-Register und APIC-Timer erfolgversprechend und wird die Emulation normaler Ethernet-Geschwindigkeit erlauben.

Darüber hinaus sollte die Realitätsnähe der Emulation durch weitergehende Messungen bestätigt werden. Zusätzlich zum Verhalten des Durchsatzes in Abhängigkeit von der Netzlast, ist die Überprüfung eines ebenso abhängigen Verhaltens der Kollisionsanzahl denkbar. Außerdem interessant ist die mittlere Verzögerung beim Zugriff auf das Medium wie sie auch in Abschnitt 4.2.2 herangezogen wird.

7.3 Schlußfolgerung

Ziel dieser Arbeit war die Entwicklung einer prototypischen Implementierung eines Emulationsverfahrens für Netze mit gemeinsamem Medium. Zu Beginn stand die Untersuchung verwandter Arbeiten. Diese führte zur Identifikation einer Klassifizierung von Emulationsverfahren für Netze im allgemeinen. Es ist festzustellen, daß zur Emulation von kabelgebundenen Netzen mit gemeinsamem Medium wie beispielsweise LANs keine bisherigen Arbeiten existieren. Insofern betritt die vorliegende Arbeit in diesem Bereich Neuland.

Vor der Erarbeitung der Konzepte wurden für deren spätere Evaluation neun möglichst objektive und von bestimmten Konzepteigenschaften unabhängige Kriterien festgelegt. Anschließend konnten Konzepte erarbeitet werden, die verschiedene der bereits identifizierten Klassen von Emulationsverfahren abdecken. Hierzu gehören Konzepte sowohl hohen als auch niedrigen Emulationsgrades sowie verteilter und zentraler Architektur. Der Emulationsgrad beschreibt dabei die Abstraktion eines Emulationsverfahrens von einer realen Sicherungsschicht-Implementierung und somit der tiefsten in Emulations-Software realisierbaren Netzschicht.

Aus der Evaluation der fünf erarbeiteten Konzepte ging die Emulation von Ethernet durch Nachbildung des Medienzugriffs in Software als erfolgversprechender Ansatz hervor. Dieses Konzept basiert auf der Annahme, daß Übertragungszeiten in der Emulationsumgebung gegenüber den durch die Emulation eingeführten Verzögerungen vernachlässigbar sind. Auf dieser Grundlage bildet das Verfahren ein verteiltes Verständnis der Signalträger auf dem emulierten, gemeinsamen Medium. Dadurch lassen sich die für Ethernet notwendigen Charakteristika Träger- und Kollisionserkennung sowie Übertragungsratenlimitierung und Übertragungsverzögerung emulieren.

Für den ausgewählten Ansatz wurde eine Implementierung in Form eines Linux-Kernel-Moduls für die NET-Umgebung, d.h. für die dort vorgesehene Kernel-Version 2.4 und Intel Pentium Prozessorfamilie, erstellt. Das entstandene sogenannte Ethemu-Modul funktioniert mit einem normalen, unveränderten Kernel, der für die entsprechende Zielarchitektur, d.h. Intel 586 oder höher, übersetzt wurde und ist somit zu den derzeit gängigen Linux-Distributionen kompatibel.

Um die Realitätsnähe des Ethemu-Konzepts zu untersuchen, wurden Messungen vorgenommen. Dabei kamen Meßreihen zum Einsatz, die das Verhalten des Durchsatzes in Abhängigkeit von der Netzlast jeweils für verschiedene emulierte Übertragungsraten aufnehmen. Die Meßergebnisse zeigen, daß Ethemu offenbar mindestens Übertragungsraten emulieren kann, die um drei Größenordnungen größer sind, als die sichere Rate, die Bit-genaue Kollisionserkennung erlaubt. Es scheint sich nicht grob negativ auszuwirken, wenn Überschneidungen nicht Bit-genau, sondern nur blockweise über eine ganze Reihe von Bits erkennbar sind. Diese Aussage ist gültig für eine Zeitgranularität von 10 Millisekunden und eine sichere Übertragungsrate von $\frac{1\,\text{Bit}}{10\,\text{ms}} = 100\,\text{Bit/s}$.

Bei einer Verwendung der vorgeschlagenen Verbesserungen für feinere Zeitgranularität verkürzt sich das kleinste meßbare Zeitintervall z.B. bei einem mit 1 GHz getakteten Prozessor auf 1 Nanosekunde, d.h. eine Verbesserung um sieben Größenordnungen. Unter Berücksichtigung, daß auch das Auslesen von Zeitstempelregistern einige Taktzyklen in Anspruch nehmen kann, stellt sich mit großzügig aufgerundeten 100 Nanosekunden Zeitgranularität immer noch eine Verbesserung um fünf Größenordnungen ein. Dann ergibt sich als sichere, emulierte Übertragungsrate 10 MBit/s. Dies erlaubt bereits die sichere Emulation von normalem Ethernet.

Unter der Annahme, daß sich die Ergebnisse bezüglich Kollisionserkennung auf die verbesserte Situation übertragen lassen, scheint auch Fast-Ethernet möglich. Allerdings kommt dann die emulierte Übertragungsrate in die Nähe der physikalischen Rate der Emulationsumgebung von 1 GBit/s. Hier wäre es möglich, daß die Übertragungszeiten in der Emulationsumgebung nicht mehr gegenüber den Emulationsverzögerungen vernachlässigbar sind. Da das Ethemu-Konzept gerade auf dieser Annahme

beruht, könnte eine einfache Übertragung der Ergebnisse auf die höhere Rate verhindert werden. Immerhin lag die Meßreihe mit der schnellsten Rate von 80 kBit/s relativ weit, nämlich ca. zwei Größenordnungen, unter der langsamsten physikalischen Verbindung der Meßumgebung von 10 MBit/s.

Das Ergebnis dieser Arbeit ist also eine funktionsfähige, verteilte Ethernet-Emulation, die mit kleinen Verbesserungen mindestens normales Ethernet mit 10 MBit/s sicher und in Echtzeit emulieren kann. Dabei ist nicht nur die Emulation kabelgebundener Netze mit gemeinsamem Medium neu, sondern auch die Art und Implementierung des Konzepts: Es wird die Medienzugriffsschicht in Software nachgebildet. Im Gegensatz zu anderen möglichen Emulationsansätzen kommen hierbei keine abstrahierenden Berechnungsmodelle zum Einsatz. Außerdem liegt die Implementierung auf niedrigster Software-Schicht im Protokollstapel und erlaubt es, Software auf allen darüber liegenden Netzschichten, d.h. auf Vermittlungs-, Transport- und Anwendungsschicht zu testen. Ich bin zuversichtlich, daß Ethemu eine Bereicherung der NET-Umgebung sein wird und dort hilfreich seine Dienste verrichtet.

Anhang A

Formeln für MAC-Kennlinien

In diesem Anhang sind die Formeln aus [Sid99] und hierzu verwendete Konstanten, die zur Gewinnung der Netzlast-Verhaltenskurven in Abb. 4.4 verwendet wurden, enthalten.

A.1 Verwendete Konstanten und abgeleitete Größen

Allgemein

Normalisierte Last: G

Normalisierter Durchsatz: S

Ankunftsrate in einer Station in Pakete/s: $\lambda = 0,5$

Anzahl Knoten: $M = 100$

Paketgröße in Bit: $L = 1500 \cdot 8$

Übertragungsrate in Bit/s: $C = 10 \cdot 10^6$

ALOHA

Zeit für erneuten Versuch in Sekunden: $B = 1,5$

Zeitdauer erfolgreicher Übertragung in Sekunden: $T = L/C$

CSMA

Contentionslot in Sekunden: $\tau = 51,2 \cdot 10^{-6}$

Normalisiertes τ: $a = \tau/T$

Faktor für Jambitdauer als Produkt aus τ und γ: $\gamma = 1$

A.2 Formeln für lastabhängigen Durchsatz der Medienzugriffsverfahren

ideal: $S(G) = G$

ALOHA $S(G) = Ge^{-2G}$

slotted ALOHA: $S(G) = Ge^{-G}$

non-persistent CSMA: $S(G,a) = \frac{Ge^{-aG}}{G(1+2a)+e^{-aG}}$

slotted non-persistent CSMA: $S(G,a) = \frac{aGe^{-aG}}{1-e^{-aG}+a}$

1-persistent CSMA: $S(G,a) = \frac{Ge^{-G(1+2a)}[1+G+aG(1+G+aG/2)]}{G(1+2a)-(1-e^{-aG})+(1+aG)e^{-G(1+a)}}$

slotted 1-persistent CSMA: $S(G,a) = \frac{Ge^{-G(1+a)}\left(1+a-e^{-aG}\right)}{(1+a)(1-e^{-aG})+ae^{-G(1+a)}}$

non-persistent CSMA/CD: $S(G,a,\gamma) = \frac{Ge^{-aG}}{Ge^{-aG}+\gamma aG(1-e^{-aG})+2aG(1-e^{-aG})+2-e^{-aG}}$

Literaturverzeichnis

[ACO97] Mark Allman, Adam Caldwell, and Shawn Ostermann. ONE: The Ohio Network Emulator. Technical Report TR-19972, Ohio University, School of Electrical Engineering and Computer Science, 18 August 1997.

[ADLY95] Jong Suk Ahn, Peter B. Danzig, Zhen Liu, and Limin Yan. Evaluation of TCP Vegas: Emulation and Experiment. *ACM Computer Communication Review*, 25(4):185–195, 1995.

[Aiv01] Tigran Aivazian. Linux Kernel 2.4 Internals, 23 August 2001. Linux Documentation Project.

[BBD+01] Michael Beck, Harald Böhme, Mirko Dziadzka, Ulrich Kunitz, Robert Magnus, Claus Schröter, and Dirk Verworner. *Linux-Kernelprogrammierung: Algorithmen und Strukturen der Version 2.4*. Addison-Wesley, 6., aktualisierte und erweiterte edition, 2001.

[Cox96] Alan Cox. Kernel Korner: Network Buffers and Memory Management. *Linux Journal*, Tuesday, 01 October 1996.

[DBCF95] Nigel Davies, Gordon S. Blair, Keith Cheverst, and Adrian Friday. A Network Emulator To Support The Development Of Adaptive Applications. In *Proceedings of the 2nd USENIX Symposium on Mobile and Location Independent Computing*, pages 47–55, Ann Arbor, Michigan, U.S.A., April 1995.

[Dud02] Dominique Dudkowski. Emulationskonzepte für Mobilfunk- und Ad-Hoc-Netze. Diplomarbeit Nr. 2004, Universität Stuttgart, Abteilung Verteilte Systeme, 2002.

[Fal99] Kevin Fall. Network Emulation in the Vint/NS Simulator. In *Proceedings of the Sixth International Symposium on Modeling, Analysis and Simulation of Computer Telecommunication Systems*, pages 244–250, Red Sea, Egypt, 1999.

[Gur99] Andrei V. Gurtov. Technical Issues of Real-Time Network Simulation in Linux. Technical Report FDPW'99 Volume 2, University of Helsinki, Department of Computer Science, 1999.

[Hen02] Bryan Henderson. Linux Loadable Kernel Module HOWTO, 21 May 2002. Linux Documentation Project.

[HLR02] Daniel Herrscher, Alexander Leonhardi, and Kurt Rothermel. Modeling Computer Networks for Emulation. In *Proceedings of the 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '02)*, pages 1725–1731, Las Vegas, June 2002.

[HMLR01] Daniel Herrscher, Christian Maihöfer, Alexander Leonhardi, and Kurt Rothermel. A Network Emulation Testbed for Performance Analysis of Distributed Applications. Internal Report, University of Stuttgart, 2001.

[HR02] Daniel Herrscher and Kurt Rothermel. A Dynamic Network Scenario Emulation Tool. In *Proceedings of the 11th International Conference on Computer Communications and Networks (ICCCN 2002)*, Miami, October 2002. (to appear).

[Ins01] Gianluca Insolvibile. Kernel Korner: The Linux Socket Filter: Sniffing Bytes over the Network. *Linux Journal*, Friday, 01 June 2001.

[Ins02] Gianluca Insolvibile. Kernel Korner: Inside the Linux Packet Filter. *Linux Journal*, Friday, 01 February 2002.

[IP94] David B. Ingham and Graham D. Parrington. Delayline: A Wide-Area Network Emulation Tool. *Computing Systems*, 7(3):313–332, 1994.

[KGM+01] Markku Kojo, Andrei Gurtov, Jukka Manner, Pasi Sarolahti, Timo Alanko, and Kimmo Raatikainen. Seawind: a Wireless Network Emulator. In *Proceedings of the 11th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems*, Aachen, Germany, September 2001.

[kl99] kossak and lifeline. Building Into The Linux Network Layer. *Phrack Magazine*, 9(11), 09 September 1999. `http://www.phrack.org/phrack/55/P55-12`.

[NIS97] NIST, Advanced Network Technologies Division. NIST Net, 1997. `http://www.antd.nist.gov/itg/nistnet/`.

[PF94] Vern Paxson and Sally Floyd. Wide-Area Traffic: The Failure of Poisson Modeling. In *Proceedings of the ACM SIGCOMM '94 Conference on Communication Architectures, Protocols, and Applications*, pages 257–268, 1994.

[PF97] Vern Paxson and Sally Floyd. Why We Don't Know How To Simulate The Internet. In *Proceedings of the 1997 Winter Simulation Conference*, Atlanta, Georgia, U.S.A., December 1997.

[Pom99] Ori Pomerantz. Linux Kernel Module Programming Guide, 26 April 1999. Version 1.1.0, Linux Documentation Project.

[RC01] Alessandro Rubini and Jonathan Corbet. *Linux Device Drivers*. O'Reilly, 2nd edition, June 2001.

[Riz97] Luigi Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, January 1997.

[RS90] Raphael Rom and Moshe Sidi. *Multiple Access Protocols: Performance and Analysis*. Springer-Verlag, New York, 1990.

[Rus99] David A. Rusling. The Linux Kernel, 25 January 1999. Review, Version 0.8-3, Linux Documentation Project.

[Sal99] Jamal Hadi Salim. Kernel Korner: IP Bandwidth Management. *Linux Journal*, Tuesday, 01 June 1999.

[Sid99] Moshe Sidi. Multiple Access Schemes. In *Encyclopedia of Electrical and Electronic Engineering*, volume 13, pages 663–674. John Wiley & Sons, 1999.

[Smo00] Miquel van Smoorenburg. Linux serial console. Source Code Documentation, Linux Kernel 2.4.18, 2000.

[T+02] Linus Torvalds et al. Linux 2.4.18. Kernel Source Code, 2002.

[Tan96] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, 3rd edition, 1996.

[Tor01] Linus Torvalds. Linux kernel coding style. Source Code Documentation, Linux Kernel 2.4.18, 2001.

[TW+01] Linus Torvalds, G. W. Wettstein, et al. Oops tracing. Source Code Documentation, Linux Kernel 2.4.18, 2001.

[Wau01] Tim Waugh. kernel-doc nano-HOWTO. Source Code Documentation, Linux Kernel 2.4.18, 2001.

[WPR+02] Klaus Wehrle, Frank Pählke, Hartmut Ritter, Daniel Müller, and Marc Bechler. *Linux Netzwerkarchitektur: Design und Implementierung von Netzwerkprotokollen im Linux-Kern*. Addison-Wesley, 2002.

[WTSW95] Walter Willinger, Murad S. Taqqu, Robert Sherman, and Daniel V. Wilson. Self-Similarity through High Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level. In *Proceedings of the ACM SIGCOMM '95 Conference on Communication Architectures, Protocols, and Applications*, pages 100–113, 1995.

[YR98] Ikjun Yeom and A. L. Narasimha Reddy. ENDE: An End-to-end Network Delay Emulator. Technical report, Texas A&M University, Department of Electrical Engineering, 1998.

[ZB93] Yongguang Zhang and Bharat Bhargava. A Facility For Experimenting Distributed Software in the Internet. In *Proceedings of the IEEE Workshop in Advances in Parallel and Distributed Systems*, pages 40–45, Princeton, New Jersey, U.S.A., October 1993.

[ZL02] Yongguang Zhang and Wei Li. An Integrated Environment for Testing Mobile Ad-Hoc Networks. In *Proceedings of the Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '02)*, EPFL Lausanne, Switzerland, June 2002.

Index

Ich versichere, daß ich diese Arbeit selbständig verfaßt und nur die angegebenen Hilfsmittel verwendet habe.

Stuttgart, den

_____

Steffen Maier