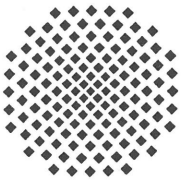


# Universität Stuttgart

Institut für Nachrichtenvermittlung und Datenverarbeitung
Prof. Dr.-Ing. habil. Dr. h.c. P. J. Kühn

**67. Bericht über verkehrstheoretische Arbeiten**

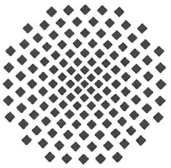
# Effizienz von Verfahren zur adaptiven und verteilten Verkehrslenkung in Paketvermittlungsnetzen

von
**Martin Lang**

1997

D 93


Druck: E. Kurz & Co., Druckerei + Reprografie GmbH., Stuttgart
ISBN 3-922403-77-8



# University of Stuttgart

Institute of Communication Networks and Computer Engineering
Prof. Dr.-Ing. habil. Dr. h.c. P. J. Kühn

**67th Report on Studies in Congestion Theory**

# Efficiency of Adaptive and Distributed Routing Algorithms in Packet Switching Networks

by
**Martin Lang**
1997

# Efficiency of Adaptive and Distributed Routing Algorithms in Packet Switching Networks

## Summary

Availabilty of information gets more and more important in our current information society. To fulfill the growing communication needs, new networks based on more powerful communication technologies are introduced. In order to protect investments, existing networks are also extended. Each change of the topology of a network has direct influence on the routing used. The major goals of this report are to develop a new routing scheme that is capable to adapt fast to a changing network topology. The need to shut down single network nodes or the whole network should be avoided. To improve reaction time on link failures the scheme should be able to provide multiple alternative paths. All paths must be loop-free.

The newly developed routing scheme is evaluated and compared to existing ones. Since the timely behavior and the distributed state of all network nodes is of special interest for these comparative studies, an interactive protocol simulator has been developed, based on an object-oriented simulation library.

## Chapter 1: Introduction

This chapter explains the necessity and importance of routing in modern communication networks and defines the detailed goals of this report.

## Chapter 2: Fundamentals

This chapter introduces the basic concepts used in this report. The first section introduces the basic principles of communication. It gives an overview of different network topologies used in communication networks and explains important terms as circuit switching, packet switching, connection-oriented communication, and connectionless communication. Finally, the reference model for open system interconnection is presented.

The second section explains the principles of object-oriented software design. Here, the basic terms of object, class, abstraction, modularisation, encapsulation and hierarchy are defined. The section ends with a short introduction to software design patterns.

The last section of this chapter gives an overview of the object-oriented simulation library used as a basis of the protocol simulator developed later in this thesis. The overall architecture and the most important concepts of the library are explained.

## Chapter 3: Routing Algorithms

In the past two decades, many different routing algorithms have been proposed. To get a better understanding of the algorithms, this chapter starts with presenting classification schemes which are generally used for classifying routing algorithms. Next, the most important algorithms used in circuit-switched networks are presented. Since this reports deals mainly with packet-switched networks, the algorithms used in those networks are explained in more detail.

One class of routing algorithms are non-adaptive routing algorithms. The main representatives of this class are static routing, bifurcated routing, flooding, and selective flooding which are presented shortly.

One of the mostly used routing schemes is the so called shortest-path routing. In this scheme each link of the network gets a length assigned to it which is based on an arbitrary metric. All traffic is routed via the shortest path, i.e. the sequence of links that has the shortest length. The shortest path for each source destination pair is calculated using iterative algorithms, e.g., Dijkstra's algorithm. The working principle of Dijkstra's algorithm is explained and illustrated by a simple example network.

As main representatives of the class of adaptive routing algorithms the "hot-potato" algorithm, the backward-learning algorithm and delta routing are presented. Adaptive routing algorithms use special protocols to support the exchange of information that is needed for the calculation of the routes. The terms "link-state protocol" and "distance-vector protocol" are introduced and explained.

The chapter concludes with a section on optimal routing.

## Chapter 4: An Object-Oriented Protocol Simulator for the Evaluation of Routing Algorithms

The major problem with development and testing of distributed routing algorithms is their distributed nature. Since those algorithms run on different network nodes in parallel, their interworking is sometimes difficult to understand and to test. A suitable tool can help to make the behaviour of those algorithms more transparent.

This chapter presents a protocol simulator that has been developed in this thesis and has been used for the development, testing and evaluation of the routing algorithm of chapter 5. The main targets for the simulator where: a user must be able to enter different network topologies, different simulation models for network nodes and links must be supported, easy integration of different routing protocols, interactive simulation execution, and, finally, should a user be able to set breakpoints during the execution of a distributed routing protocol or run it in single-step mode.

The simulator was developed using object-oriented design techniques. It is based on the simulation library presented in chapter 2. After the principle architecture and mechanisms of the simulator have been presented, the software design patterns are explained which are used during the development.

## Chapter 5: A New Adaptive Routing Algorithm for Packet-Switched Networks

This is the main chapter of the report. Here, a new distributed, adaptive routing procedure is developed and optimized. The procedure consists of three main components: a shortest-path algorithm that is able to calculate several alternative and loop-free paths which are sorted by their lenghts; a component that selects one of these paths for each new connection or packet (in case of connectionless communication); the third component is a special routing protocol that provides the necessary information for the algorithm.

The developed routing protocol is based on a distance-vector protocol. Each network node sends a distance vector to each of its neighbors which contains the length of the selected path and its transfer nodes for each destination node. The path information that is submitted to the neighbor nodes belongs always to the actual path that is selected by the second component. Since this selection may vary, depending on the neighbor from which a packet is received, the distance vectors are calculated individually for each neighbor. Based on this information, the routing algorithm is able to calculate a number

of paths that differ in their first link, to eliminate those out that contain loops, and to sort them by their lengths. The calculated paths are stored in a routing table.

Finally, two optimizations of the introduced algorithm and protocol are presented. One optimizes the amount of path information that needs to be transmitted in the distance vectors, the second optimizes the routing algorithm in order to provide more loop-free alternative paths.

## Chapter 6: Comparative Study of Different Routing Schemes

This chapter presents the results of a comparative study of different routing algorithms. The various versions (no optimization, optimized distance vectors, optimized number of alternatives) of the routing scheme presented in chapter 5 are compared to an ordinary distributed Bellman-Ford routing algorithm and a link-state protocol using Dijkstra's algorithm for path calculation. For the evaluations the topology of multiple typical networks was changed on a link-by-link basis. The behavior of the protocols was observed using the protocol simulator presented in chapter 4. To get representative results, statistics have been gathered over the single events.

It is shown in that the routing scheme developed in this thesis has advantages over the traditional distance-vector or link-state protocols. In case of changing network topologies, the number of temporary loops in paths is significantly reduced. The shortest paths are always loop-free.

The availability of a network is also improved. With the developed routing scheme, less source – destination pairs are affected by a change in the network topology. If one destination node cannot be reached, the duration of this disconnected state is in general shorter than with traditional routing algorithms.

## Chapter 7: Summary and Outlook

This chapter summarizes the report and gives a short outlook for future studies.

The presented routing algorithm is based on an arbitrary metric that is used to define the length of a link. One issue that this report does not deal with is the definition of a suitable metric for various networks and load situations. Another area of interest could be the definition of a scheme that uses the calculated alternative paths for load sharing.

# Inhaltsverzeichnis

**Abkürzungen** 5

**Formelzeichen** 7

**1 Einleitung** 8

1.1 Kommunikationsnetze . . . . . 8

1.2 Ziele der Arbeit . . . . . 10

1.3 Übersicht über die Arbeit . . . . . 10

**2 Grundlagen** 12

2.1 Grundlagen der Kommunikation . . . . . 12

2.1.1 Netztopologien . . . . . 12

2.1.2 Vermittlungsverfahren . . . . . 13

2.1.2.1 Durchschaltevermittlung . . . . . 13

2.1.2.2 Paketvermittlung . . . . . 14

2.1.2.3 Weitere Vermittlungprinzipien . . . . . 14

2.1.3 Verbindungskonzepte . . . . . 15

2.1.4 Verkehrslenkung . . . . . 17

2.1.5 Das OSI-Referenzmodell der offenen Kommunikation . . . . . 18

2.2 Objektorientierter Softwareentwurf . . . . . 20

2.2.1 Grundlagen des objektorientierten Softwareentwurfs . . . . . 21

2.2.1.1 Objekte . . . . . 21

2.2.1.2 Klassen . . . . . 22
2.2.1.3 Abstraktion . . . . . 22
2.2.1.4 Modularisierung . . . . . 23
2.2.1.5 Datenkapselung . . . . . 23
2.2.1.6 Hierarchie . . . . . 23
2.2.2 Softwareentwurfsmuster . . . . . 24
2.3 Objektorientierte Simulation . . . . . 26
2.3.1 Architektur der Simulationsbibliothek . . . . . 27
2.3.2 Modellkomponenten . . . . . 28
2.3.3 Verbindung von Modellkomponenten . . . . . 30
2.3.4 Ereignisbearbeitung . . . . . 32
2.3.5 Simulationssteuerung . . . . . 33

**3 Verkehrslenkungsalgorithmen** **34**

3.1 Allgemeines Klassifizierungsschema . . . . . 35
3.2 Algorithmen für die Durchschaltevermittlung . . . . . 38
3.3 Algorithmen für die Paketvermittlung . . . . . 39
3.3.1 Nicht-adaptive Verkehrslenkung . . . . . 39
3.3.2 Shortest Path Routing . . . . . 40
3.3.3 Adaptive Verkehrslenkung . . . . . 42
3.3.4 Optimale Verkehrslenkung . . . . . 46
3.3.5 Sonstige Verfahren . . . . . 51

**4 Ein objektorientierter Protokollsimulator zur Untersuchung von Verkehrslenkungsalgorithmen** **52**

4.1 Anforderungen . . . . . 52
4.2 Aufbau des Protokollsimulators . . . . . 53
4.3 Modellierung . . . . . 55

4.4 Flexible Eingabe der Netztopologie . . . . . 59
4.5 Objektorientierte Dekomposition . . . . . 63
4.5.1 Übertragung von Steuerinformation . . . . . 63
4.5.2 Algorithmen . . . . . 65
4.5.3 Knoten- und Leitungsmodelle . . . . . 66
4.6 Ein allgemeines Konzept zur Überwachung des Protokollablaufs . . . . . 71
4.6.1 Motivation . . . . . 71
4.6.2 Realisierung . . . . . 73
4.7 Wiederverwendbare Entwurfsmuster . . . . . 75
4.7.1 Objektfabrik . . . . . 76
4.7.2 Prototyp . . . . . 77
4.7.3 Globale Instanzen . . . . . 78

**5 Ein neues Verfahren zur adaptiven Verkehrslenkung in Paketvermittlungsnetzen 81**

5.1 Anforderungen an eine moderne Verkehrslenkung . . . . . 81
5.2 Ein adaptives Verfahren mit Alternativwegen . . . . . 82
5.2.1 Grundlegender Aufbau . . . . . 82
5.2.2 Ungelöste Probleme . . . . . 86
5.2.3 Ähnliche Ansätze . . . . . 88
5.3 Realisierung mit Link-State-Algorithmen . . . . . 89
5.4 Entwurf eines neuen Protokolls . . . . . 91
5.4.1 Wegeauswahl . . . . . 92
5.4.2 Ein einfaches Update-Protokoll . . . . . 93
5.4.2.1 Berechnung der Distanzvektoren . . . . . 93
5.4.2.1.1 Struktur eines Distanzvektors . . . . . 93
5.4.2.1.2 Individuelle Berechnung . . . . . 94
5.4.2.1.3 Berechnungszeitpunkt . . . . . 96

5.4.2.2 Versenden eines Distanzvektors . . . . . 97
5.4.3 Berechnung mehrerer Alternativen . . . . . 97
5.5 Optimierung des Protokolls . . . . . 99
5.5.1 Optimierung der Verkehrslenkungstabellen . . . . . 99
5.5.2 Reduzierung der Meldungsgröße . . . . . 102

**6 Vergleich verschiedener Verfahren zur Verkehrslenkung 107**
6.1 Konfiguration und Effizienzkriterien . . . . . 107
6.2 Bewertung . . . . . 109
6.2.1 Schleifenfreiheit . . . . . 109
6.2.2 Meldungsaustausch . . . . . 110
6.2.3 Erreichbarkeit . . . . . 112
6.2.4 Protokolloverhead . . . . . 115
6.2.5 Adaption an Lastschwankungen . . . . . 117

**7 Zusammenfassung und Ausblick 118**
7.1 Zusammenfassung . . . . . 118
7.2 Ausblick . . . . . 120

**Literaturverzeichnis 122**

**Anhang**

**A Notation für OOD-Diagramme 134**
A.1 Klassendiagramme . . . . . 134
A.2 Objektdiagramme . . . . . 136

# Abkürzungen

| | |
|---|---|
| ARPA | Advanced Research Projects Agency |
| ARPANET | Advanced Research Projects Agency Network |
| ATM | Asynchronous Transfer Mode |
| CCITT | Comité Consultatif International Téléphonique et Télégraphique |
| CS | Durchschaltevermittlung (circuit switching) |
| CSMA/CD | Carrier Sense Multiple Access / Collision Detection |
| FCS | Schnelle Durchschaltevermittlung (fast circuit switching) |
| FDDI | Fiber Distributed Data Interface |
| FPS | Schnelle Paketvermittlung (fast packet switching) |
| HS | Hybridvermittlung (hybrid switching) |
| ISDN | Dienstintegrierendes digitales Netz (integrated services digital network) |
| ISO | International Organization for Standardization |
| ITU | International Telecommunication Union |
| ITU-T | Telecommunication Standardization Sector of ITU |
| LAN | Lokales Netz (local area network) |
| MAN | Regionales Hochgeschwindigkeitsnetz (metropolitan area network) |
| MCCS | Mehrkanaldurchschaltevermittlung (multi channel circuit switching) |
| MFDL | Pfad mit geringstem Kostenanstieg (minimal first derivative length) |
| OOA | Objektorientierte Analyse (object-oriented analysis) |
| OOD | Objektorientierter Entwurf (object-oriented design) |
| OOP | Objektorientierte Programmierung (object-oriented programming) |
| OSI | Open Systems Interconnection |
| PBX | Nebenstellenanlage (private branch exchange) |
| PCI | Protocol control information |
| PCM | Pulscodemudulation |
| PS | Paketvermittlung (packet switching) |
| SAP | Dienstzugangspunkt (service access point) |
| WAN | Weitverkehrsnetz (wide area network) |

# Verkehrslenkungsalgorithmen

| | |
|---|---|
| BF | Bellman-Ford Algorithmus |
| DA | Algorithmus nach Kapitel 5.4 (distributed with alternatives) |
| LS | Link-State Algorithmus |
| OPR | Algorithmus mit Optimierungen nach Kapitel 5.5.1 und 5.5.2 (optimized and path reduced) |
| OPT | Algorithmus mit Optimierung nach Kapitel 5.5.1 (optimized) |
| PR | Algorithmus mit Optimierung nach Kapitel 5.5.2 (path reduced) |

# Formelzeichen

| | |
|---|---|
| $(i,j)$ | Kante zwischen den Knoten $i$ und $j$ |
| $\alpha$ | Iterationsschrittweite |
| $\alpha^{(\nu)}$ | Iterationsschrittweite des $\nu$-ten Iterationsschritts |
| $C$ | Kapazität |
| $C_{ij}$ | Kapazität der Kante $(i,j)$ |
| $d_{ij}$ | Länge bzw. Kosten der Kante $(i,j)$ |
| $D$ | Gesamtkosten des Netzes |
| $D_{ij}$ | Länge bzw. Kosten des Weges von Knoten $i$ zu $j$ |
| $D_{ij}^{(h)}$ | Länge des Weges von Knoten $i$ zu $j$, wobei der Weg maximal $h$ Leitungen enthalten darf |
| $D_{ij,k}$ | Länge des Weges von Knoten $i$ zu $j$, der über den Nachbarn $k$ von $i$ führt |
| $F_{ij}$ | Fluß über die Kante $(i,j)$ |
| $\overline{F}_{ij}$ | Fluß über die Kante $(i,j)$, der sich aus den Wegflüssen $\overline{x}_p$ ergibt |
| $L_p$ | Menge der Kanten $(i,j)$, die zu $p$ oder zu $\overline{p}$ gehören, nicht aber zu beiden |
| $M_{Tracepoint}$ | Menge der Überwachungsebenen eines Kontrollpunktes |
| $M_{Enity}$ | Menge der Überwachungsebenen einer Modellkomponente |
| $M_{Packet}$ | Mege der Überwachungsebenen eines Pakets |
| $N$ | Anzahl der Knoten eines Netzes |
| $p$ | Weg (Pfad) |
| $\overline{p}_w$ | Pfad mit geringstem Kostenanstieg (MFDL-Pfad) des Quelle-Ziel-Paars $w$ |
| $P_w$ | Menge der Wege, die das Quelle-Ziel-Paar $w$ verbinden |
| $r_w$ | Eingangsfluß des Quelle-Ziel-Paars $w$ |
| $w$ | Quelle-Ziel-Paar |
| $W$ | Menge aller Quelle-Ziel-Paare |
| $x_p$ | Fluß über den Weg $p$ |
| $\overline{x}_p$ | Fluß über den Weg $p$, der sich ergibt,wenn alle $r_w$ ausschließlich über den zugehörigen MFDL-Pfad $\overline{p}_w$ gelenkt werden |
| $\vec{x}$ | Vektor aller Wegflüsse $x_p$ |
| $\vec{x}^{(\nu)}$ | Flußvektor des $\nu$-ten Iterationsschritts |
| $\vec{\overline{x}}$ | Flußvektor aller $\overline{x}_p$ |

# Kapitel 1

# Einleitung

## 1.1 Kommunikationsnetze

Die moderne Gesellschaft ist geprägt durch eine Vielzahl verschiedenster Daten und Informationen, die von Ihren Mitgliedern gesammelt und verarbeitet werden müssen. Die Verfügbarkeit von Informationen wird zu einem zentralen Kriterium von Erfolg und Mißerfolg. Im Laufe des 20. Jahrhunderts wurden zunehmend Möglichkeiten entwickelt, Informationen auf elektrischem oder optischem Wege auch über große Entfernungen hinweg über Kommunikationsnetze auszutauschen. Anfänglich dienten die Netze hauptsächlich der Sprachübertragung. In einer modernen Informationsgesellschaft ist aber gerade die Übermittlung von Daten von großer Bedeutung. Der ständig steigende Bedarf an Datenkommunikation führte zur Entwicklung und Einführung von speziellen Datennetzen. Für den Benutzer stellt sich ein Kommunikationsnetz in Form von verschiedenen Kommunikationsdiensten dar, die durch das Netz erbracht werden. Beispiele für solche Kommunikationsdienste sind Fernsprechen (Telefon), Datenkommunikation (Datex), Fernschreiben (Telex), Bürofernschreiben (Teletex), Fernkopieren (Telefax), Bildschirmtext (BTX) und andere. Nachdem in der Anfangsphase häufig separate Netze für die verschiedenen Kommunikationsdienste bereitgestellt wurden, zeigt sich heutzutage die Tendenz zu diensteintegrierenden Netzen wie ISDN (integrated services digital network).

Je nach geographischer Ausdehnung unterscheidet man Weitverkehrsnetze (wide area networks, WANs), Regionale Hochgeschwindigkeitsnetze (metropolitan area networks, MANs), Lokale Netze (local area networks, LANs) und Nebenstellenanlagen (private branch exchanges, PBXs). Im Gegensatz zu Weitverkehrsnetzen, die in aller Regel öffentliche Netze sind und von nationalen Postverwaltungen betrieben werden, sind Lokale Netze und Nebenstellenanlagen in privater Hand. Sie dürfen den Privatgrund, auf dem sie installiert sind (z.B. Firmengelände oder Universitäts-Campus), nicht verlassen. Nebenstellenanlagen sind vorwiegend für die Sprachübertragung und Lokale Netze für die

Datenübertragung geeignet. Metropolitan Area Networks trifft man sowohl im öffentlichen als auch im privaten Bereich an. Sie arbeiten mit höherer Übertragungsgeschwindigkeit als Lokale Netze (typischerweise im Bereich von 100 Mbit/s im Gegensatz zu 10 Mbit/s von Lokalen Netzen). Metropolitan Area Networks eignen sich dazu, verschiedene lokale Kommunikationsnetze auch über größere Entfernungen miteinander zu verbinden.

Für die Vermittlung der Informationen innerhalb eines Kommunikationsnetzes gibt es zwei Hauptvermittlungsverfahren, die Durchschaltevermittlung oder Leitungsvermittlung (circuit switching, CS) und die Paketvermittlung (packet switching, PS). Bei der Durchschaltevermittlung erhält jede Verbindung zwischen zwei Kommunikationsteilnehmern einen durchgeschalteten Übertragungsweg. Dieser Übertragungsweg steht beiden Teilnehmern für die gesamte Dauer der Kommunikationsverbindung exklusiv zur Verfügung. Bei der Paketvermittlung wird die zu übertragende Information in kleine Blöcke, sogenannte Pakete, unterteilt und zwischen den Kommunikationspartnern verschickt. Die Durchschaltevermittlung ist für Dienste geeignet, die eine feste Bandbreite erfordern. Für die Integration von Diensten mit stark unterschiedlichen Bandbreiteanforderungen oder reine Datendienste ist die Paketvermittlung besser geeignet. Aus diesem Grunde basieren moderne Datennetze auf diesem Vermittlungsprinzip.

Sollen Informationen zwischen zwei Kommunikationspartnern über ein Kommunikationsnetz ausgetauscht werden, hat das Netz unter anderem die Aufgabe, einen Weg von einem Partner durch das Netz zu dem anderen Partner zu finden. Die Bestimmung eines solchen Wegs bezeichnet man als Verkehrslenkung. Sie erfolgt bei durchschaltevermittelnden Netzen beim Aufbau der Kommunikationsbeziehung, bei paketvermittelnden Netzen entweder beim Aufbau der Kommunikationsbeziehung oder für jedes Paket individuell. Die Leistungsfähigkeit eines Netzes wird zu einem wesentlichen Teil durch die Verkehrslenkung mitbestimmt. Adaptive Verfahren sind in der Lage, sich an verändernde Verhältnisse innerhalb des Netzes anzupassen. Durch die Anpassung an sich ändernde Verkehrsverhältnisse können unter Umständen Engpässe innerhalb des Netzes umgangen und so die Transferzeiten reduziert werden.

Bei einem ständig steigenden Kommunikationsbedarf und dem Wunsch, Daten direkt zwischen Datenverarbeitungsanlagen auszutauschen, gelangen vorhandene Datennetze schnell an ihre Kapazitätsgrenze. Die Einführung neuer Netztechnologien ist ein Weg, die Probleme zu lösen. Um bereits getätigte Investitionen zu schützen und um die Übergangsphase, bis neue Technologien für die Teilnehmer zu akzeptablen Preisen angeboten werden können, zu überbrücken, werden vorhandene Netze auch weiter ausgebaut.

Jede Änderung der Topologie eines Netzes hat direkten Einfluß auf die Verkehrslenkung. Eine solche Änderung kann beabsichtigt sein, weil das Netz erweitert wird, oder unbeabsichtigt durch Störungen, Ausfälle oder kurzzeitige Unterbrechung von Übertragungswegen zustande kommen. Ein gutes Verkehrslenkungsverfahren sollte jede Änderung

in der Topologie des Netzes selbständig erkennen und die Wege entsprechend anpassen. Da die Verfügbarkeit des Netzes für den Teilnehmer ein wesentliches Gütekriterium darstellt, sollte die Zeit, die das Verkehrslenkungsverfahren benötigt, um sich an eine neue Topologie anzupassen, so kurz wie möglich sein und das Netz während dieser Zeit auf alle Fälle betriebsbereit bleiben.

Die Probleme einer adaptiven Verkehrslenkung sind in der Literatur schon seit längerem bekannt (siehe z.B. [101, 102, 103, 104]). Technologische Entwicklungen auf der Seite des Netzequipments und zunehmende Teilnehmerzahlen eröffnen jedoch neue Möglichkeiten und stellen neue Anforderungen, wie z.B. die verteilte Berechnung und die Bereitstellung mehrerer Alternativen aus Sicherheits- und Zuverlässigkeitsgründen. Ferner sollten die Knoten auch bei stark veränderlicher Netztopologie immer erreichbar bleiben.

## 1.2 Ziele der Arbeit

Das Hauptziel der Arbeit ist es, ein Verkehrslenkungsverfahren zu entwerfen, das in der Lage ist, schnell und flexibel auf Änderungen in der Netztopologie zu reagieren. Das Netz soll dabei zu jedem Zeitpunkt betriebsbereit bleiben. Während der Zeit, in der sich die Verkehrslenkung an die neuen Verhältnisse im Netz anpaßt, sollen Schleifen in den Wegen vermieden werden. Die Bestimmung der Wege soll verteilt erfolgen und mehrere, nach ihrer Länge sortierte Alternativen liefern. Die Eigenschaften des entworfenen Verfahrens sollen untersucht und mit bestehenden verglichen werden.

Auf Grund des verteilten Ablaufs sind verteilte Verkehrslenkungsverfahren oft schwerer verständlich und nachvollziehbar. Um die Entwicklung und Untersuchung speziell von verteilten Algorithmen zu erleichtern, soll ein geeignetes Werkzeug entwickelt werden. Ein wesentlicher Gesichtspunkt beim Entwurf des Werkzeugs ist die Berücksichtigung verschiedener Netztopologien. Die Bestimmung der Wege soll mit Hilfe des Werkzeugs schrittweise nachvollziehbar sein. Außerdem muß das Werkzeug universell für verschiedene Verkehrslenkungsverfahren einsetzbar und auch für zukünftige Anforderungen leicht erweiterbar sein.

## 1.3 Übersicht über die Arbeit

Im folgenden Kapitel 2 werden zunächst die allgemeinen Grundlagen der Kommunikation behandelt. Es wird das OSI-Referenzmodell der offenen Kommunikation kurz vorgestellt. Weiterhin werden dort verschiedene Vermittlungsprinzipien sowie Verbindungskonzepte erläutert. Da sich das entwickelte Werkzeug der objektorientierten Simulationstechnik

bedient, wird in Kapitel 2 auch auf die Methodik des objektorientierten Softwareentwurfs, auf Softwareentwurfsmuster sowie auf die objektorientierte Simulation eingegangen.

In Kapitel 3 wird zuerst ein allgemeines Klassifizierungsschema für Verkehrslenkungsalgorithmen vorgestellt. Anschließend erfolgt eine Darstellung der aus der Literatur bekannten Verfahren. Die gängigsten Verfahren für die Durchschalte- und die Paketvermittlung werden hier kurz erläutert.

Kapitel 4 beschäftigt sich mit dem Werkzeug, mit dessen Hilfe das neue Verfahren entwickelt und mit den bestehenden Algorithmen verglichen wurde. Zu Beginn des Kapitels werden allgemeine Anforderungen an ein solches Werkzeug definiert. Anschließend werden die Konzepte erläutert, die einen flexiblen Aufbau verschiedener Netztopologien, die Überwachung des Protokollablaufs sowie die Untersuchung verschiedener Verkehrslenkungsverfahren ermöglichen. Abschließend werden allgemeine, auch für andere Zwecke wiederverwendbare Entwurfsmuster, herausgearbeitet.

Zu Beginn des Kapitels 5 werden die Anforderungen an ein neues Verkehrslenkungsverfahren zusammengestellt. Es wird eine Architektur vorgestellt, die unter anderem Komponenten enthält, die für die Wegeauswahl beim Verbindungsaufbau bzw. bei der Vermittlung von Paketen, die Bestimmung der Wege und die Verteilung von Verkehrslenkungsdaten verantwortlich sind. Für diese Komponenten werden verschiedene Realisierungsmöglichkeiten vorgestellt und diskutiert. Schließlich wird ein neues Protokoll entwickelt, das die verteilte Berechnung mehrerer Alternativwege erlaubt. Abschließend wird dieses Protokoll schrittweise optimiert.

In Kapitel 6 werden die Eigenschaften des neuen Verfahrens untersucht und mit bekannten verglichen.

Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick.

# Kapitel 2

# Grundlagen

## 2.1 Grundlagen der Kommunikation

In diesem Kapitel sollen einige Grundbegriffe der Kommunikationstechnik erläutert werden, die für die Arbeit von Bedeutung sind.

### 2.1.1 Netztopologien

Kommunikationsnetze können unterschiedliche Topologien besitzen. Man kann fünf verschiedene elementare Strukturen unterscheiden, die in Reinform nur in kleineren Netzen auftreten. Diese elementaren Topologien sind:

- **Ring**: Einige Medienzugriffsprotokolle für Lokale Netze (z.B. der Token Ring [76]) oder Metropolitan Area Networks (z.B. FDDI [77]) setzen eine Ringstruktur voraus.
- **Bus**: Viele Lokale Netze, wie z.B. CSMA/CD [74] oder der Token Bus [75], haben eine Busstruktur.
- **Stern**: Sternförmige Topologien findet man häufig in Nebenstellenanlagen oder im Teilnehmerzugangsbereich größerer Netze.
- **Maschenform**: Bei den maschenförmigen Netzen unterscheidet man vollständige und unvollständige Vermaschung. Die Zentralvermittlungsstellen des deutschen Telefonnetzes sind z.B. vollständig vermascht.
- **Baum**: Baumförmige Strukturen findet man vereinzelt bei Lokalen Netzen [64] oder Zugangsnetzen [23].

Bei größeren Netzen treten diese elementaren Strukturen selten in Reinform auf. Dort findet man eher Mischformen. Große Netze sind oft hierarchisch aufgebaut, wobei auf

jeder Ebene eine andere Topologie sinnvoll sein kann. Höhere Hierarchieebenen haben in der Regel weniger Knoten und weisen häufig eine Maschen- oder Ringform auf. Niedere Ebenen bestehen dagegen meist aus vielen Knoten, die sternförmig verbunden sind.

### 2.1.2 Vermittlungsverfahren

Bei den Vermittlungsverfahren kann man zwei grundsätzliche Prinzipien unterscheiden: die *Durchschaltevermittlung* (circuit switching, CS) und die *Teilstreckenvermittlung*, auch *Speichervermittlung* genannt (store and forward switching).

Bei der Durchschaltevermittlung wird für die gesamte Dauer einer Kommunikationsbeziehung ein durchgehender Übertragungsweg konstanter Bandbreite zur Verfügung gestellt. Dieser Übertragungsweg steht beiden Teilnehmern exklusiv zur Verfügung. Die Bereitstellung des Übertragungswegs ist unabhängig davon, ob Informationen übertragen werden oder nicht.

Kennzeichnend für die Speichervermittlung ist, daß es keinen durchgeschalteten Übertragungsweg gibt. Die Informationen werden bei diesem Vermittlungsprinzip blockweise zusammengefaßt und mit einem Kopf versehen, in dem Steuerinformation, entweder als vollständige Zieladresse oder in Form einer virtuellen Kanalnummer, eingetragen ist. Anhand dieser Steuerinformation wird die Nachricht durch das Netz transportiert. Kann die Nachricht in einem Netzknoten nicht sofort vermittelt werden, wird sie dort kurzzeitig zwischengespeichert. Der Zugriff auf einen Übertragungskanal erfolgt durch statistisches Multiplexen.

#### 2.1.2.1 Durchschaltevermittlung

Wie bereits oben kurz erwähnt wurde, wird bei der Durchschaltevermittlung beiden Teilnehmern ein durchgehender Übertragungsweg zur exklusiven Nutzung bereitgestellt. Früher wurde pro Verbindung (siehe Abschnitt 2.1.3) eine Leitung durchgeschaltet. Man bezeichnet dies dann auch als *Leitungsvermittlung*.

Durch den Einsatz der Zeitmultiplextechnik ist es heute möglich, eine Leitung für mehrere Verbindungen gleichzeitig zu nutzen. Grundlage hierfür ist ein Pulsrahmen konstanter Länge (z.B. PCM-Rahmen mit 125$\mu$s), der in eine feste Anzahl von Kanälen unterteilt wird. Die Übertragungskapazität (Bitrate, oft auch nicht ganz korrekt als „Bandbreite“ bezeichnet) eines Kanals läßt sich aus der Rahmendauer und der Anzahl der Kanäle pro Rahmen bestimmen. Einer Verbindung steht nun nicht mehr eine ganze Leitung sondern nur ein oder mehrere Zeitkanäle zur Verfügung. Man spricht in diesem Zusammenhang auch von *Kanalvermittlung*.

#### 2.1.2.2 Paketvermittlung

Die *Paketvermittlung* (packet switching, PS) ist die wichtigste Form der Speichervermittlung. Bei ihr wird die zu übertragende Nachricht in mehrere Blöcke mit begrenzter Länge unterteilt. Jeder dieser Nachrichtenblöcke wird getrennt mit einem Kopf versehen, in dem die Steuerinformation eingetragen ist. Die Nachrichtenblöcke werden dann entweder unabhängig voneinander anhand der expliziten Zieladresse oder entlang eines vorher aufgebauten Pfads anhand der dafür zugeteilten virtuellen Kanalnummer durch das Netz transportiert.

Die Vorteile der Paketvermittlungstechnik sind:

- die Mehrfachausnutzung von Verbindungs- und Anschlußleitungen;
- die Erhöhung der Fehlersicherheit durch spezielle Fehlererkennungs- und Fehlerkorrekturmechanismen;
- die Zusammenarbeit von Endgeräten mit unterschiedlicher Geschwindigkeit durch Einsatz von Flußkontrollverfahren;
- die Zusammenarbeit von Endgeräten mit unterschiedlichen Übertragungsverfahren durch Einsatz von Anpassungsfunktionen innerhalb des Netzes oder am Netzzugang;
- die einfache Integration neuer Dienste.

#### 2.1.2.3 Weitere Vermittlungprinzipien

Neben den beiden Hauptvermittlungsprinzipien, der Durchschaltevermittlung und der Paketvermittlung, gibt es noch weitere Verfahren, die hier ebenfalls kurz erläutert werden sollen.

Normalerweise wird bei der Durchschaltevermittlung immer nur ein Kanal pro Verbindung durchgeschaltet. Sie ist deshalb nicht besonders gut für die Integration verschiedener Dienste mit unterschiedlichen Bandbreiteanforderungen geeignet. Bei der *Mehrkanal-Durchschaltevermittlung* (multi channel circuit switching, MCCS) können einer Verbindung mehrere Kanäle zugeordnet werden. Auf diese Art lassen sich Verbindungen realisieren, deren Bandbreitebedarf ein ganzzahliges Vielfaches der Bandbreite eines CS-Grundkanals ist.

Bei der *schnellen Durchschaltevermittlung* (fast circuit switching, FCS) wird einer Verbindung nur dann ein durchgeschalteter Übertragungsweg zur Verfügung gestellt, wenn tatsächlich auch Nachrichten übertragen werden. Zwischen jeweils zwei Übertragungen werden die Betriebsmittel für andere Verbindungen freigegeben. Auf diese Weise läßt sich eine bessere Auslastung der Netzressourcen erreichen.

Eine andere Form der Teilstreckenvermittlung ist die *Sendungsvermittlung* (message switching). Bei dieser Vermittlungsart wird die Nachricht nicht wie bei der Paketvermittlung in mehrere Blöcke unterteilt, sondern als Ganzes auf einmal übertragen.

Die *schnelle Paketvermittlung* (fast packet switching, FPS) ist eine Weiterentwicklung der konventionellen Paketvermittlung. Die Protokollfunktionalität wurde reduziert, so daß eine rein hardwaremäßige Bearbeitung möglich ist [4, 149].

Es existieren auch *hybride Vermittlungstechniken* (hybrid switching, HS, CS/PS), die ebenfalls auf einem Pulsrahmen basieren, der in Zeitschlitze konstanter Länge aufgeteilt wird. Die Vermittlung von CS und PS-Verkehr erfolgt durch geeignete Zuteilung der Zeitschlitze [66].

Der beim zukünftigen Breitband-ISDN zum Einsatz kommende *Asynchronous Transfer Mode* (ATM) [60, 114] ist vom Prinzip her paketvermittelnd. Die Nachrichten werden mittels Zellen konstanter Länge (5 Oktetts Zellkopf plus 48 Oktetts Information) übertragen. Im Gegensatz zur klassischen Paketvermittlung wird aber bereits beim Aufbau einer Verbindung (siehe Abschnitt 2.1.3) sichergestellt, daß die Verbindung die geforderte „Bandbreite" erhält und Dienstgüteparameter (z.B. Zellverlustwahrscheinlichkeit) eingehalten werden. Kann dies nicht garantiert werden, wird ein entsprechender Verbindungswunsch abgelehnt.

### 2.1.3 Verbindungskonzepte

Eine Kommunikationsbeziehung zwischen zwei oder mehreren Funktionseinheiten, sogenannten Instanzen, eines Kommunikationsnetzes wird als *Verbindung* bezeichnet. Die Kommunikationsbeziehung kann temporär oder dauerhaft sein. Eine Verbindung gliedert sich in drei unterschiedliche Phasen:

- **Verbindungsaufbau**

  Durch den Aufbau einer Verbindung wird ein Kanal zwischen Sender und Empfänger reserviert. Alle für das jeweilige Vermittlungsprinzip benötigten Ressourcen des Netzes werden während des Verbindungsaufbaus reserviert bzw. bereitgestellt.

- **Datenübertragung**

  Während dieser Phase findet der eigentliche Austausch von Informationen statt.

- **Verbindungsabbau**

  Die während des Verbindungsaufbaus reservierten Ressourcen werden wieder freigegeben.

Prinzipiell kann man zwischen einer *verbindungsorientierten Kommunikation* und einer *verbindungslosen Kommunikation* unterscheiden. Durchschaltevermittelnde Netze arbeiten verbindungsorientiert. Der durchgeschaltete Übertragungsweg stellt dabei eine sogenannte *physikalische Verbindung* dar.

Bei paketvermittelnden Netzen kennt man sowohl die verbindungsorientierte als auch die verbindungslose Kommunikation. Bei der verbindungslosen Kommunikation, die auch als *Datagramm-Betrieb* bezeichnet wird, werden einzelne Pakete durch das Netz transportiert. Besteht eine Nachricht aus mehreren Paketen, werden alle Pakete unabhängig voneinander übertragen. Es werden im Netz keine Betriebsmittel reserviert, so daß es zu Paketverlusten kommen kann. Die Pakete können verschiedene Wege durch das Netz

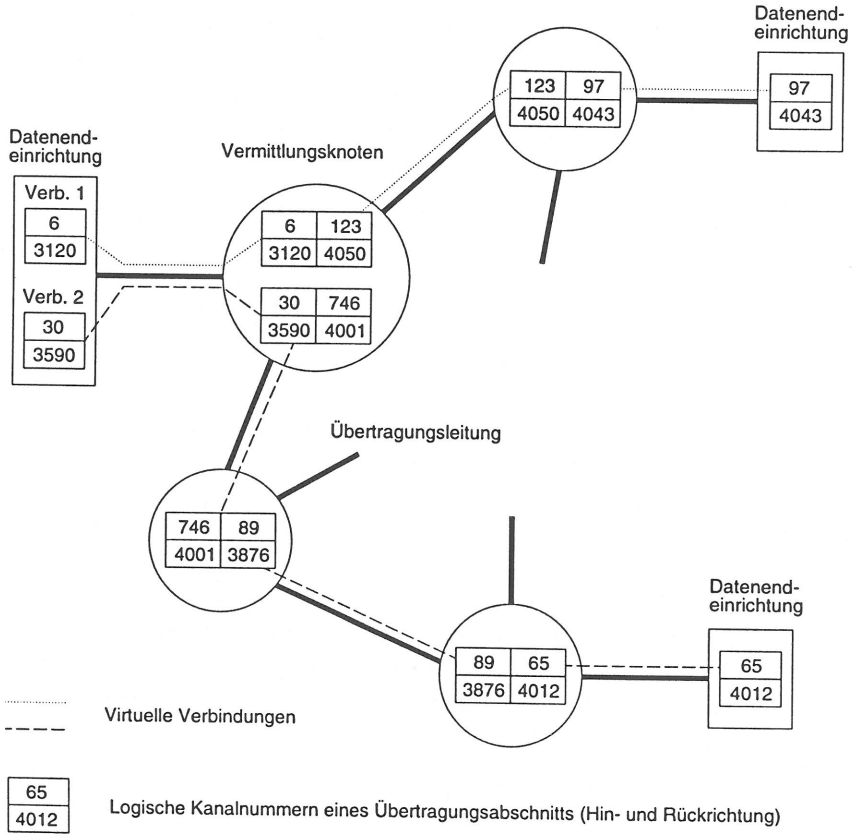



Bild 2.1: Prinzip der virtuellen Verbindung

nehmen, weshalb die Reihenfolge, in der die Pakete beim Empfänger ankommen, nicht garantiert werden kann.

Die verbindungsorientierte Kommunikation in paketvermittelnden Netzen basiert auf dem Prinzip der *virtuellen Verbindung*. Es ist in Abbildung 2.1 dargestellt.

Bei einer virtuellen Verbindung werden auf jedem Übertragungsabschnitt logische Kanäle gebildet und mit logischen Kanalnummern versehen. Völlig analog zu einer physikalischen Verbindung, die als Kettung physikalischer Kanäle angesehen werden kann, ist die virtuelle Verbindung die Aneinanderreihung logischer Kanäle. Die Verkettung der logischen Kanäle erfolgt innerhalb der Vermittlungsknoten durch Tabellen. Pakete, die zwischen zwei Teilnehmern über eine virtuelle Verbindung ausgetauscht werden, müssen nicht mehr die vollständige Zielinformation mit sich führen. Es genügt eine eindeutige Identifikation der Verbindung. Da alle Pakete einer virtuellen Verbindung den gleichen Weg durch das Netz nehmen, kann die ursprüngliche Reihenfolge beim Empfänger mit Hilfe von Fehlersicherungs- und Fehlerbehebungsmaßnahmen garantiert werden. Paketverluste infolge von Fehlern oder Pufferüberläufen können durch wiederholte Übertragung vermieden werden.

### 2.1.4 Verkehrslenkung

Bevor Nachrichten zwischen zwei Kommunikationspartnern über ein Netz ausgetauscht werden können, muß ein Weg vom Sender durch das Netz zum Empfänger gefunden werden. In aller Regel soll der Verkehr über möglichst kurze Wege geführt werden, bzw. die Laufzeit durch das Netz soll so gering wie möglich sein. Es ist die Aufgabe der Verkehrslenkung (*Routing*), solche Wege für jedes Teilnehmerpaar (Quelle-Ziel-Paar) des Netzes zu finden.

Bei der verbindungsorientierten Kommunikation findet die Verkehrslenkung beim Aufbau einer Verbindung statt. Die Zieladresse des Empfängers befindet sich entweder in einem speziellen Verbindungsaufbaupaket (Paketvermittlungsnetze) oder wird einem Vermittlungsknoten über eine Signalisierung (Durchschaltevermittlung) mitgeteilt. Anhand dieser Zieladresse bestimmt der Vermittlungsknoten den nächsten Wegabschnitt. Bei der Durchschaltevermittlung kann die Verbindung jetzt durchgeschaltet werden. Paketvermittlungsknoten speichern eine Kennung der Verbindung zusammen mit dem berechneten Wegabschnitt in einer Tabelle, der sogenannten Verbindungstabelle, ab. Nachfolgende Datenpakete müssen nun nicht mehr die vollständige Empfängeradresse mit sich führen. Es genügt die Kennung der Verbindung. Durch einfaches Nachschlagen in der Verbindungstabelle kann das Paket auf den nächsten Übertragungsabschnitt vermittelt werden.

Bei der verbindungslosen Kommunikation trägt jedes Paket die vollständige Zieladresse des Empfängers mit sich. Die Verkehrslenkung erfolgt für jedes Paket getrennt anhand

der Zieladresse. Da sich manche Verkehrslenkungsverfahren dynamisch an die Belastung des Netzes oder einzelner Übertragungsabschnitte anpassen, ist nicht gewährleistet, daß zwei aufeinanderfolgende Pakete mit derselben Zieladresse den gleichen Weg nehmen.

Für die Bestimmung der Wege gibt es verschiedene Verkehrslenkungsverfahren bzw. -algorithmen. Eine ausführliche Diskussion der Verfahren sowie eine Klassifizierung erfolgt in Kapitel 3.

### 2.1.5 Das OSI-Referenzmodell der offenen Kommunikation

Um die Komplexität eines Kommunikationssystems überschaubarer zu machen, sind die meisten Netz- bzw. Protokollarchitekturen geschichtet aufgebaut. Der Grundgedanke dabei ist, daß jede Schicht einen ganz bestimmten Dienst für die nächsthöhere Schicht erbringt. Sie stellt diesen Dienst unter Ausnutzung der Dienste, die sie ihrerseits von der nächsttieferen Schicht angeboten bekommt, zur Verfügung. Implementierungsdetails bleiben der höheren Schicht verborgen. Die Anzahl der Schichten kann von Architektur zu Architektur variieren.

Nach mehrjähriger Arbeit wurde von der **International Organization for Standardization** (ISO) 1984 das *Basisreferenzmodell für die Verbindung offener Systeme* (Open Systems Interconnection, OSI) standardisiert [69]. Das Referenzmodell definiert sieben Schichten, die jeweils unabhängig von der darunterliegenden Schicht sind. Jede Schicht bietet der darüberliegenden Schicht ihre Dienste über *Dienstzugangspunkte* (service access points, SAPs) an. Die Kommunikation zwischen zwei benachbarten Schichten erfolgt über *Dienstprimitive*, die an den Dienstzugangspunkten ausgetauscht werden (vgl. [69]).

Im Referenzmodell werden zwei Arten von Systemen unterschieden, *Endsysteme* und *Transitsysteme*. *Verarbeitungsinstanzen*, d.h. die Einheiten, zwischen denen die Kommunikation letztendlich stattfindet, sind nur in Endsystemen vorhanden. Transitsysteme verbinden Endsysteme, sofern diese nicht direkt miteinander verbunden sind. Die verschiedenen Schichten des Referenzmodells sind in Abbildung 2.2 dargestellt. Endsysteme enthalten alle sieben Schichten, in Transitsystemen sind nur die untersten drei Schichten vorhanden. Die Schichten 1–4 werden häufig auch als *Transportsystem* bezeichnet, die Schichten 5–7 als *Verarbeitungssystem*.

In der *Bitübertragungsschicht* (physical layer) werden die mechanischen, elektrischen oder optischen Eigenschaften des Mediums und die Darstellung der Bits auf dem Medium festgelegt. Sie ist ferner für eine bitweise Übertragung der Nachrichten verantwortlich.

Die wesentliche Aufgabe der *Sicherungsschicht* (data link layer) ist es, eine gesicherte, abschnittsweise Nachrichtenübertragung zur Verfügung zu stellen. In Lokalen Netzen ist das Medienzugriffsprotokoll ebenfalls in der Schicht 2 angeordnet.

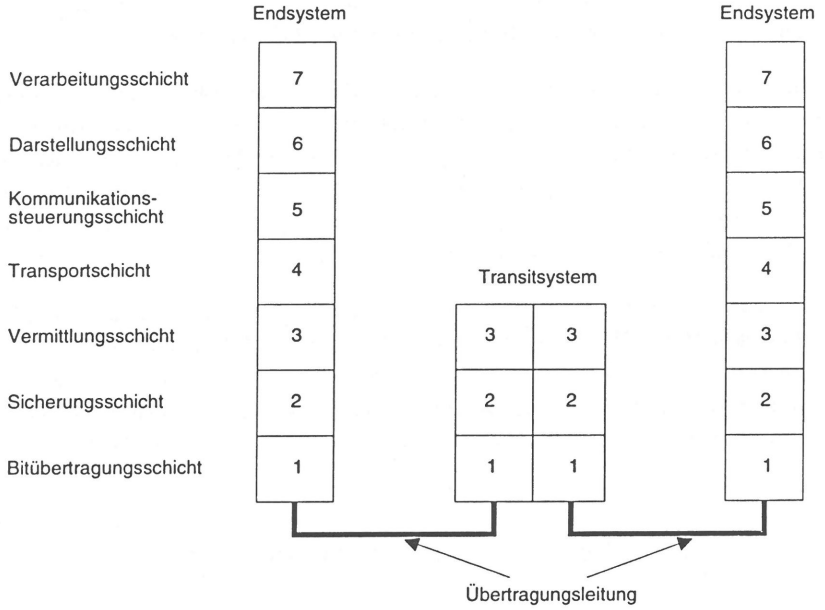



Bild 2.2: Das Basisreferenzmodell zur Kommunikation zwischen offenen Systemen

Schicht 3 ist die *Vermittlungsschicht* (network layer). Sie ist die oberste Schicht eines Transitsystems, d.h. eines Netzes. Die Aufgaben der Vermittlungsschicht sind die Bereitstellung aller Mittel, die zum Aufbau, zur Unterhaltung und zum Abbau von Netzwerkverbindungen zwischen Endsystemen sowie zum Austausch von Daten über diese Verbindungen notwendig sind. Als wesentlicher Teil gehört dazu auch die Verkehrslenkung. Die Vermittlungsschicht soll die darüberliegende Transportschicht von allen Fragen, die die Verkehrslenkung und die Übertragung über das Netz betreffen, abschirmen. Aus der Sicht der Transportschicht muß sich das Netz als eine Art transparenter Übertragungskanal darstellen.

Die *Transportschicht* (transport layer) garantiert einen zuverlässigen Transport von Paketen vom Sender zum Empfänger unter Einhaltung einer geforderten Dienstgüte. Sie ist die unterste Schicht, die eine Ende-zu-Ende Kommunikation ermöglicht.

Die Aufgabe der *Kommunikationssteuerungsschicht* (session layer) ist die Dialogsteuerung. Dazu gehören z.B. die Vergabe von Sendetoken, d.h. die Festlegung, wer als nächstes senden darf, und die Definition von Wiederaufsetzpunkten nach Fehlerfällen.

Die *Darstellungsschicht* (presentation layer) sorgt für eine Angleichung verschiedener Datendarstellungen. Abschließend bietet Schicht 7, die *Verarbeitungsschicht* (application

layer), verschiedene Dienstelemente zur Anwendungsunterstützung an. Eine ausführliche Beschreibung der Aufgaben aller Schichten ist in [126, 145] enthalten.

Im Laufe der Jahre wurden von verschiedenen Gremien [59, 144] Standards für die einzelnen Schichten erarbeitet und von ISO verabschiedet. Auf Grund der unterschiedlichen Anforderungen an die Kommunikation gibt es für jede Schicht nicht genau einen Standard. Es existieren in der Regel mehrere Standards pro Schicht oder zumindest mehrere Klassen eines Standards, die den jeweiligen Anforderungen am besten gerecht werden. Für die Vermittlungsschicht sind dies z.B. [71, 72, 73].

Für öffentliche Paketvermittlungsnetze sind die Protokolle und Funktionen der Vermittlungsschicht in ITU-T X.25 (ehemals CCITT X.25) [70, 79] festgeschrieben. Der Standard beschreibt die Schnittstelle am Netzzugang zwischen der *Datenendeinrichtung* und der *Datenübertragungseinrichtung*. Die entsprechende Schnittstelle zwischen den Vermittlungsknoten innerhalb des Netzes ist in ITU-T X.75 (ehemals CCITT X.75) [80] definiert.

## 2.2 Objektorientierter Softwareentwurf

Komplexität ist nach Brooks [24] eine inherente Eigenschaft von Softwaresystemen. Um diese Komplexität zu meistern, werden Softwaresysteme in kleinere, verständlichere Teile zerlegt. In der Vergangenheit wurde dazu hauptsächlich die algorithmische Dekomposition verwendet, bei der jedes Modul einen wesentlichen Teil im Gesamtablauf des Systems darstellt (siehe z.B. [98, 162]). Eine weitere Dekompositionsmethode, die in jüngster Zeit zunehmend an Bedeutung gewinnt, ist die objektorientierte Dekomposition. Bevor auf die wesentlichen Elemente der Objektorientierung näher eingegangen wird, sollen zuerst einige Begriffe erläutert werden.

Der Entwurf großer Softwaresysteme erfolgt in der Regel in mehreren Phasen. Während der Analysephase werden die Anforderungen an das Softwaresystem im Problembereich identifiziert. Man erhält ein Modell, das alle relevanten Aspekte des realen Systems abdeckt. In der anschließenden Entwurfsphase wird dieses Modell in eine abstrakte Lösung umgesetzt, wobei spezifische Randbedingungen, wie Kosten, Laufzeit, Eigenschaften des Zielsystems, etc. mit berücksichtigt werden. Abschließend wird diese Lösung implementiert und getestet.

Beim objektorientierten Ansatz bezeichnet man diese Phasen mit *objektorientierter Analyse* (object-oriented analysis, OOA), *objektorientiertem Entwurf* (object-oriented design, OOD) und mit *objektorientierter Programmierung* (object-oriented programming, OOP).

### 2.2.1 Grundlagen des objektorientierten Softwareentwurfs

Jede Softwareentwurfsmethode basiert auf ihren eigenen konzeptionellen Grundgedanken. Über die Elemente, die ein Ansatz unterstützen muß, um als objektorientiert zu gelten, herrscht noch kein Konsens. Meyer [106] macht die Objektorientierung vom Vorhandensein der folgenden Elemente abhängig:

- Modularisierung,
- Datenkapselung,
- Klassen,
- Vererbung,
- Polymorphie.

Fehlt die Vererbung, spricht man häufig von objektbasiert anstatt von objektorientiert [25]. Da *Klassen* und *Objekte* bei beiden Ansätzen eine zentrale Rolle spielen, definiert Booch ein Objektmodell [20], das als Grundlage sowohl für die objektorientierte als auch für die objektbasierte Programmierung dient. Die vier wesentlichen Bestandteile dieses Modells sind:

- Abstraktion,
- Datenkapselung,
- Modularisierung,
- Hierarchie.

#### 2.2.1.1 Objekte

In einem objektorientierten Programm wird die Funktion durch Zusammenarbeit vieler Objekte erbracht. Jedes Objekt besitzt interne Datenstrukturen (*Felder*) und Operationen (*Methoden*), die auf das Objekt angewendet werden können. Es existiert dabei ein sehr enger Verbund zwischen den Daten eines Objekts und seinen Operationen. Die Daten können in der Regel nicht direkt sondern nur durch die Operationen des Objekts verändert werden. Jedes Objekt besitzt einen *Zustand*, der durch die Summe der statischen Eigenschaften (welche Felder existieren, welchen Typ besitzen diese, etc.) und den normalerweise dynamischen Werten dieser Felder gekennzeichnet ist.

Objekte tauschen Meldungen untereinander aus. Erhält ein Objekt eine Meldung, reagiert es, indem es bestimmte Operationen ausführt. Das *Verhalten* eines Objektes definiert, welche Meldungen empfangen werden können und wie auf den Empfang der jeweiligen Meldungen reagiert wird. Schließlich besitzt jedes Objekt eine eindeutige *Identität*, die es von allen anderen Objekten unterscheidet.

#### 2.2.1.2 Klassen

Viele Objekte weisen gemeinsame Eigenschaften auf, d.h. sie haben gleiche Datenfelder und definieren dieselben Operationen. Eine Klasse ist eine Menge von Objekten, die die gleiche interne Struktur und dasselbe Verhalten haben. Jedes Objekt ist die Instanz einer Klasse. Obwohl zwei Objekte derselben Klasse die gleiche interne Datenstruktur aufweisen und dieselben Operationen definieren, bedeutet dies nicht, daß sie auch einen identischen Zustand haben bzw. die Operationen das gleiche Ergebnis liefern. Als Beispiel kann man sich eine Klasse IMMOBILIE vorstellen, die Felder für die Größe, Anzahl der Zimmer, etc. und eine Operation Mietpreis definiert. Der Inhalt der Felder unterscheidet sich von Mietobjekt zu Mietobjekt, und die Operation Mietpreis liefert je nach Objekt unterschiedliche Ergebnisse.

Im Gegensatz zu Objekten, die eine eindeutige Identität besitzen und eine genau definierte Rolle innerhalb des Systems spielen, repräsentiert die Klasse die Gemeinsamkeiten aller ihrer Instanzen. Sie stellt eine Art Bindeglied zwischen einer Abstraktion (vgl. Abschnitt 2.2.1.3) und allen Objekten dar, die diese Abstraktion verkörpern. Eine Klasse definiert alle Operationen, die auf ein Objekt ausgeführt werden können, und damit auch die Schnittstelle zwischen den Objekten.

Zwischen Klassen gibt es verschiedene Beziehungen. Die *Vererbung* ist dabei die wichtigste. Sie wird in Abschnitt 2.2.1.6 gesondert besprochen. Um ihre eigene Aufgabe zu erfüllen, kann sich eine Klasse der Hilfe anderer Klassen bedienen. Man kann sich dies als eine Art Auftragsvergabe vorstellen [107]. Um den eigenen Auftrag zu erfüllen, wird er in Teilaufträge zerlegt, die wiederum an andere Klassen weitergereicht werden. Zwischen den Klassen besteht in diesem Fall eine *Benutzen-Beziehung* (using relation). Besteht zwischen zwei Objekten eine strukturelle Abhängigkeit, d.h. enthält ein Objekt ein anderes, dann besteht zwischen den zugehörigen Klassen eine *Enthaltensein-Beziehung* (containment relation).

#### 2.2.1.3 Abstraktion

*Abstraktion* ist eine elementare Möglichkeit, mit der Komplexität von Systemen umzugehen. Sie ist eine vereinfachte Beschreibung des Systems oder von Teilen daraus, die sich auf die wesentlichen Eigenschaften konzentriert und unwichtige Details vernachlässigt. Eine gute Abstraktion konzentriert sich auf diejenigen Details, die für die Lösung des Problems wichtig sind, und vernachlässigt andere, die zumindest für den Moment unbedeutend sind.

Eine Abstraktion ist die charakteristische Eigenschaft eines Objekts, die es eindeutig von allen anderen Arten von Objekten unterscheidet. Sie kann als eine konzeptionelle Abgrenzung eines Objektes gegenüber anderen aufgefaßt werden. Dabei ist zu beachten,

daß diese Abgrenzung nicht allgemeingültig ist und durchaus vom jeweiligen Betrachter abhängt. Durch die Konzentration auf die für einen Betrachter wesentlichen Dinge trägt die Abstraktion dazu bei, das grundsätzliche Verhalten einen Objektes von dessen Implementierung zu trennen.

Man kann verschiedene Arten von Abstraktionen unterscheiden. Objekte können u.a. Abstraktionen eines sinnvollen Elements aus dem Problem- oder Lösungsbereich sein (*entity abstraction*), einen allgemeinen Satz von gleichartigen Funktionen zur Verfügung stellen (*action abstraction*) oder verschiedene Operationen zusammenfassen, die von einer übergeordneten Steuerung benötigt werden (*virtual machine abstraction*).

#### 2.2.1.4 Modularisierung

Ein wichtiges Element zur Strukturierung eines Systems stellt die Zerlegung in einzelne *Module* dar. Jedes Modul löst dabei eine oder mehrere Teilaufgaben. Die Kopplung zwischen den verschiedenen Modulen eines Systems sollte so gering wie möglich sein. Jedes Modul stellt dem Anwender eine klar definierte Schnittstelle von Diensten zur Verfügung, die von ihm erbracht werden. Die Implementierung bleibt dem Anwender verborgen. Die Modularisierung ist ein wichtiges Strukturierungskonzept, das man auch bei nicht objektorientierten Programmiersprachen wie Modula-2 antrifft.

#### 2.2.1.5 Datenkapselung

Die *Datenkapselung* ist eine Verfeinerung der Modularisierung. Hier werden Datenstrukturen und Operationen zu einer Einheit zusammengefaßt. Die Daten können nicht mehr direkt verändert werden. Die Operationen bilden eine klar definierte Schnittstelle zum Anwender hin. Die internen Abläufe bleiben dem Anwender verborgen. Man spricht in diesem Zusammenhang auch von *information hiding*.

#### 2.2.1.6 Hierarchie

Abstraktion und Modularisierung tragen wesentlich dazu bei, die Komplexität eines Systems zu reduzieren und somit das Verständnis zu erleichtern. Abstraktionen sind aber selten unabhängig voneinander, sondern bilden häufig eine Hierarchie. In Softwaresystemen kann man zwei unterschiedliche Formen von Hierarchie unterscheiden, die *Klassenhierarchie* und die *Objekthierarchie*.

Wie bereits im Abschnitt 2.2.1.2 erwähnt wurde, ist die Vererbung die wichtigste Beziehung zwischen Klassen, die es erlaubt, diese in Hierarchien anzuordnen. Vergleicht man

verschiedene Klassen, so stellt man fest, daß viele dieselben Operationen definieren oder die gleichen Datenfelder besitzen. Um diese Gemeinsamkeiten nicht mehrfach definieren zu müssen, bietet es sich an, eine Hierarchie zu bilden, in der die Gemeinsamkeiten in einer Oberklasse zusammengefaßt werden, und diese an andere Klassen zu *vererben*. Abgeleitete Klassen können Operationen verändern oder weitere Felder und Operationen hinzufügen. Da eine abgeleitete Klasse somit eine Spezialisierung der Oberklasse darstellt, können auf diese Weise Spezialisierungen des Problembereichs sehr einfach auf den Programmentwurf übertragen werden.

Besitzt eine Klasse genau eine Oberklasse, so spricht man von *Einfachvererbung* (single inheritance). Darf eine abgeleitete Klasse von mehreren Oberklassen erben, bezeichnet man dies als *Mehrfachvererbung* (multiple inheritance). Klassen, die selbst keine Oberklasse besitzen, von denen aber abgeleitet wird, bezeichnet man als *Basisklassen*.

Durch die Vererbung ist sichergestellt, daß eine abgeleitete Klasse alle Felder und Operationen ihrer Oberklasse besitzt. Aus diesem Grund ist es möglich, daß innerhalb eines Programms ein Objekt der Oberklasse durch ein Objekt der abgeleiteten Klasse ersetzt wird. Die Kommunikation zwischen Objekten findet durch Austausch von Meldungen statt (vgl. Abschnitt 2.2.1.1). Bindet man die Operationen, die beim Empfang der Meldung ausgeführt werden, nicht statisch beim Übersetzen eines Programms, sondern macht sie vom tatsächlichen Objekttyp abhängig — sog. *dynamisches Binden* (dynamic binding oder late binding) — so wird dadurch *Polymorphie* (Vielgestaltigkeit) möglich. Eine Immobilienfirma führt z.B. eine Liste von Objekten der Oberklasse IMMOBILIE. In diese Liste werden aber nur Objekte von den abgeleiteten Klassen WOHNUNG, EINFAMILIENHAUS, MEHRFAMILIENHAUS, etc. eingetragen. Sollen die gesamten Mieterträge berechnet werden, wird an jedes Objekt in der Liste die Meldung Mietpreis geschickt. Wird die Meldung statisch an die Operation der Oberklasse IMMOBILIE gebunden, erhält man nicht das gewünschte Ergebnis. Die Meldung muß in diesem Fall dynamisch zur Laufzeit an die Operation der jeweiligen abgeleiteten Klasse gebunden werden.

Zwischen Objekten kann eine strukturelle Abhängigkeit bestehen, d.h. ein Objekt hat nicht nur elementare Datenfelder, sondern enthält weitere Objekte. Durch diese Abhängigkeiten entsteht eine Enthaltensein-Hierarchie zwischen den einzelnen Objekten. Dies ist die zweite Form von Hierarchie, die sogenannte Objekthierarchie, in einem Softwaresystem.

### 2.2.2 Softwareentwurfsmuster

Einer der Vorteile des objektorientierten Entwurfs liegt darin, daß dieselben Abstraktionen sowohl während der Analyse- als auch während der Entwurfsphase verwendet werden

können. Das eigentliche Problem besteht jetzt darin, die richtigen Abstraktionen, Klassen und Objekte zu finden. In großen Softwaresystemen können viele wichtige Entwurfsgesichtspunkte nicht in einer einzigen Klasse erfaßt werden. Sie sind über mehrere Klassen verteilt, die in bestimmter Beziehung zueinander stehen und gemeinsam eine Aufgabe erbringen.

Es stellt sich nun die Frage: „Worin unterscheidet sich ein guter Entwurf von einem schlechten?“ Coplien schreibt in [36], daß gute Designer ein Gefühl für die Struktur einer Software haben. Sie verwenden spezielle Muster zur Lösung bestimmter Probleme. Diese Muster werden nicht nur innerhalb eines Programms mehrfach verwendet, sondern bieten allgemeinere Lösungen für grundsätzliche, immer wiederkehrende Probleme. Booch spricht in diesem Zusammenhang von Mechanismen [20] und beschreibt damit eine Softwarestruktur, bei der mehrere Objekte gemeinsam ein Verhalten aufzeigen, das zur Lösung eines isolierten Problems beiträgt.

Die Arbeiten auf dem Gebiet der *Softwareentwurfsmuster* wurden stark von den Gedanken von Alexander [3] beeinflußt, der Entwurfsmuster erstmals für die architektonische Planung von Gebäuden und Städten eingesetzt hat. Architektur bezeichnet nicht nur das Aussehen oder die Struktur eines Gebäudes, sondern hat auch ästhetische Komponenten. Alexander hat verschiedene Lösungen für architektonische Grundprobleme erstellt, die von Architekten als eine Art Baukasten verwendet werden können, um Gebäude, Siedlungen oder ganze Städte zu entwerfen. Bei der Erstellung der Muster wurden nicht nur rein bautechnische Randbedingungen berücksichtigt, sondern auch anthropologische Fragen, wie z.B. die Anordnung der Küche relativ zum Hauseingang oder eine sinnvolle, dem Lebensrhythmus entsprechende Anordnung der Räume innerhalb eines Gebäudes.

Beim Entwurf von Softwaresystemen sind Parallelen zur Architektur festzustellen. Auch hier findet man immer wieder allgemeine Grundprobleme, wie z.B. die Erzeugung von Objekten unter bestimmten Randbedingungen, das Iterieren durch verschiedene Kontainer, die Kapselung von Algorithmen und Strategien, die Anpassung von Schnittstellen, etc. Wenn sich allgemeingültige Lösungen, die nicht von einem speziellen Anwendungskontext abhängig sind, für solche Probleme finden lassen, können Softwareentwickler diese wiederverwenden und — völlig analog zu einem Architekten — ihre Programme baukastenartig zusammensetzen.

Solche allgemeinen Lösungen für Grundprobleme werden als Softwareentwurfsmuster (design patterns) bezeichnet. Sie ermöglichen die Wiederverwendung von Software auf einer höheren Abstraktionsebene als die der Klassen. Softwareentwurfsmuster sind nicht mit Ideomen gleichzusetzen. Ideome [35] bieten zwar ebenfalls Lösungen für spezielle Programmierprobleme, befinden sich aber auf einer niedrigeren Abstraktionsebene. Sie beschreiben mehr den typischen Gebrauch einer Programmiersprache oder von Sprachkonstrukten, um spezielle Programmieraufgaben effizient zu lösen [37].

Um Entwurfsmuster wiederverwenden zu können, müssen sie dokumentiert werden. In [36, 47, 50] wurden verschiedene formalisierte Beschreibungsformen für Entwurfsmuster vorgestellt. Für jedes Muster werden dort eine Definition des Problems, das es löst, eine Beschreibung des Kontexts und der Randbedingungen sowie die eigentliche Lösung dokumentiert. In [50] werden zu jedem Muster noch weitere Elemente, wie Anwendungsbeispiele, mögliche Implementierungen, Konsequenzen aus der Anwendung des Musters, verwandte Muster, u.a. aufgeführt.

## 2.3 Objektorientierte Simulation

Verteilte Systeme sind teilweise so komplex, daß sie nur sehr schwer in ihrer Gesamtheit zu verstehen sind. Um verschiedene Entwurfsalternativen bewerten und miteinander vergleichen zu können, müssen Leistungsuntersuchungen durchgeführt werden. Dadurch können kritische Punkte des Systems frühzeitig erkannt und behoben werden. Übersteigt die Komplexität eines Systems eine gewisse Schranke, gelangen analytische Untersuchungsverfahren an ihre Grenzen. Als Ausweg bleibt die Computersimulation des Systems. Bei diesem Verfahren werden die wesentlichen Eigenschaften des realen Systems durch ein Modell nachgebildet. Dieses Modell wird durch ein Computerprogramm simuliert, wobei die interessanten Kenngrößen des Systems gemessen werden.

Je nach Anwendungsfall sind verschiedene Simulationstechniken bekannt [92, 111]. Für die Simulation von Kommunikationsnetzen eignet sich besonders die Methode der zeitdiskreten, *ereignisorientierten Simulation*. Bei dieser Simulationstechnik wird das zu simulierende System nur zu Zeitpunkten betrachtet, an denen sich sein Zustand ändert. Jeder Zustandsänderung wird ein sogenanntes *Ereignis*, ein Ereigniszeitpunkt und eine Ereignisroutine, die das Ereignis bearbeitet, zugeordnet. Zeitlich ausgedehnte Vorgänge werden auf diese Weise zu einer Abfolge von Ereignissen reduziert. Die Zeitspannen zwischen jeweils zwei Ereignissen werden übersprungen, so daß sie keine Rechenzeit beanspruchen. Die eigentliche Rechenzeit wird während der Abarbeitung der Ereignisroutinen verbraucht. Die Systemzeit bleibt dabei unverändert.

Um die kausalen Zusammenhänge des realen Systems zu erhalten, müssen die Ereignisse nach ihren Ereigniszeitpunkten sortiert werden. Diese Aufgabe wird vom sogenannten Kalender übernommen. Die Ablaufsteuerung des Programmes entnimmt dem Kalender immer das Ereignis mit dem kleinsten Ereigniszeitpunkt, setzt die Systemzeit auf diesen Zeitpunkt und ruft die entsprechende Ereignisroutine auf. Durch die Bearbeitung der Ereignisroutine wird der Systemzustand verändert und in aller Regel ein neues Ereignis generiert. Auf diese Weise wird die Simulation am Leben gehalten. Während des Ablaufs der Simulation werden wichtige Kenngrößen stichprobenartig gemessen und gesammelt.

Die Simulation ist beendet, wenn genügend Meßwerte für die gewünschte statistische Aussagesicherheit erfaßt wurden.

In [87, 88] wurde eine objektorientierte Simulationsbibliothek für die ereignisorientierte Simulation entworfen. Da sie die Grundlage für das Werkzeug in Kapitel 4 darstellt, sollen die wesentlichen Elemente nachfolgend kurz vorgestellt werden.

### 2.3.1 Architektur der Simulationsbibliothek

Die Architektur der Simulationsbibliothek ist in Abbildung 2.3 dargestellt. Sie besteht im wesentlichen aus zwei Teilen. Der Teil *Simulationsunterstützung* enthält alle Komponenten, die für den Ablauf, die Steuerung und die Verwaltung eines Simulationsprogramms sowie die Messung und Auswertung von Kenngrößen notwendig sind. Der zweite Teil der Bibliothek ist das eigentliche Simulationsmodell, das hierarchisch aufgebaut sein kann.

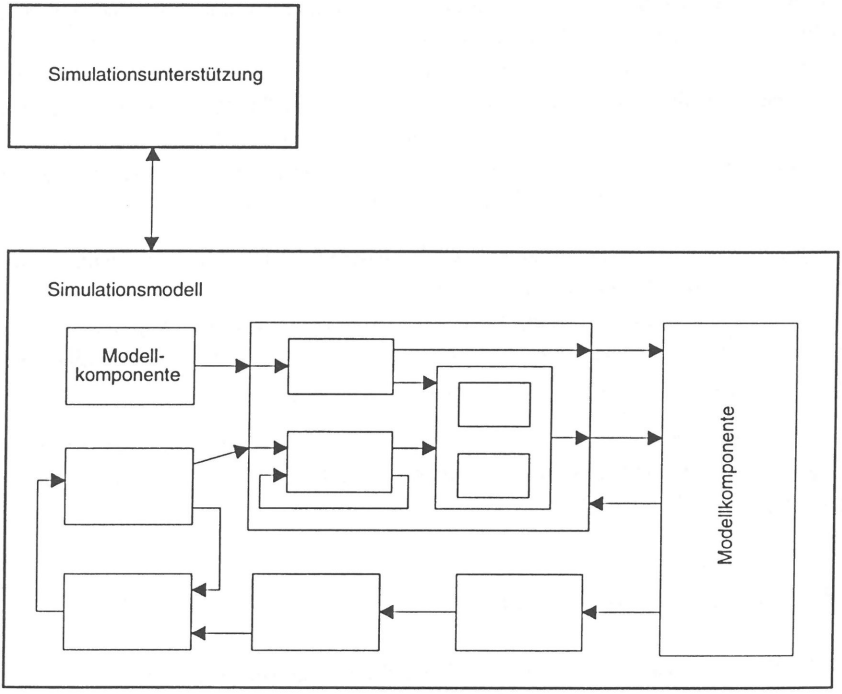



Bild 2.3: Architektur der Simulationsbibliothek

Die Kommunikation zwischen den einzelnen Modellkomponenten eines Simulationsmodells erfolgt durch Austausch von Nachrichten. Der Nachrichtenaustausch erfolgt un-

abhängig von Typ und Inhalt der Nachrichten. Die Bibliothek stellt dem Anwender eine allgemeine Nachrichtenbasisklasse zur Verfügung. Benötigt der Anwender Nachrichten mit besonderem Inhalt, kann er eine eigene Klasse von dieser Basisklasse ableiten und den Inhalt in speziellen Modellkomponenten auswerten.

Modellkomponenten können beliebig miteinander verbunden werden. Sie besitzen eine genau definierte Schnittstelle zur Außenwelt, auf die in Abschnitt 2.3.3 genauer eingegangen wird. Der Nachrichtenaustausch zwischen den Modellkomponenten wird über ein spezielles Handshake-Protokoll abgewickelt. Eine Modellkomponente stellt sich nach außen hin als „Black-Box" dar, deren Verhalten eindeutig durch die Schnittstelle definiert ist. Durch diese Trennung der strukturellen Eigenschaften einer Modellkomponente von ihrer Funktion innerhalb des Modells wird eine große Flexibilität erreicht. Eine Komponente kann beliebig aus dem Modell herausgenommen und durch eine andere ersetzt werden, ohne das Umfeld dadurch zu beeinflussen.

Ereignisse dienen bei der ereignisorientierten Simulation hauptsächlich der Steuerung des Simulationsablaufs. Sie können aber auch als Mittel zur Kommunikation zwischen Modellkomponenten aufgefaßt werden. Nachdem ein neues Ereignis erzeugt wurde, wird es an eine Modellkomponente zur Bearbeitung übergeben. Die Komponente kann dann entscheiden, ob sie das Ereignis selbst bearbeitet oder an eine übergeordnete Komponente weiterreicht. Das Simulationsmodell ist eine spezielle Modellkomponente, die den Ereigniskalender enthält. Abschnitt 2.3.4 enthält mehr über die Ereignissteuerung.

Der Block *Simulationsunterstützung* aus Abbildung 2.3 faßt alle Komponenten zusammen, die den Anwender bei der Entwicklung eines Simulationsprogramms unterstützen. Dazu gehören eine Ablaufsteuerung (vgl. Abschnitt 2.3.5 ), Unterstützung beim Einlesen von Parametern und der Ausgabe von Ergebnissen, die Erzeugung von Zufallszahlen sowie die Erfassung und statistische Auswertung von Meßwerten.

Auf die wichtigsten Konzepte der Bibliothek wird in den nachfolgenden Abschnitten genauer eingegangen. Für eine vollständige und detaillierte Beschreibung sei auf [87] verwiesen.

### 2.3.2 Modellkomponenten

Wie bereits mehrfach erwähnt wurde, stellt die Einführung von Hierarchien ein wesentliches Mittel zur Reduktion der Komplexität von Systemen dar. Die Simulationsbibliothek unterstützt diese Strukturierung, indem sie einen hierarchischen Aufbau des Simulationsmodells ermöglicht. Dadurch wird eine einfache Zuordnung der realen Objekte zu den jeweiligen Modellkomponenten erreicht. Die Abbildungen 2.4 und 2.5 zeigen ein Beispiel,

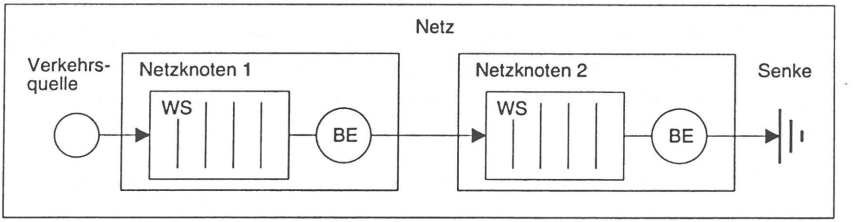



Bild 2.4: Beispiel eines Simulationsmodells

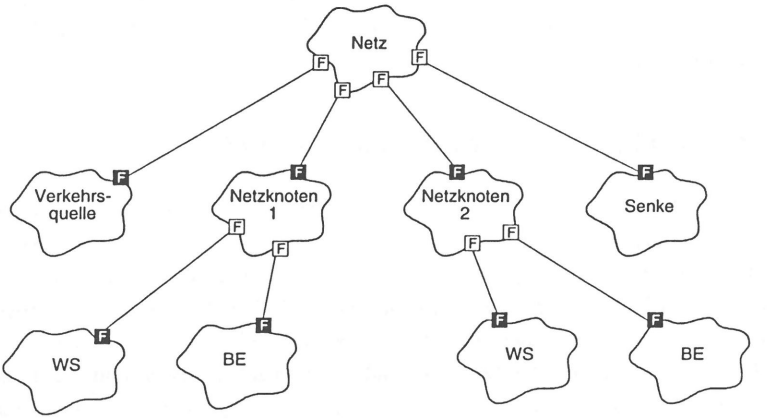



Bild 2.5: Objekthierarchie des Beispielmodells

wie ein hierarchisches Warteschlangenmodell in eine entsprechende Objekthierarchie umgesetzt werden kann.

Die grundlegenden Eigenschaften von Modellkomponenten werden in der Basisklasse TENTITY festgelegt. Modellkomponenten besitzen einen ebenfalls hierarchisch aufgebauten Namen, der ihre eindeutige Identifikation innerhalb eines Modells ermöglicht. Modellkomponenten können beliebig miteinander verbunden werden. Dies wird durch eine definierte Schnittstelle, die sogenannten Ports, ermöglicht. Die Anzahl der Ports einer Modellkomponente hängt von ihrem Typ und ihrer Funktion ab. Warteschlangen und Bedieneinheiten haben z.B. jeweils einen Eingang und Ausgang, während Multiplexer mehrere Eingänge besitzen. Ports können auch dynamisch zur Laufzeit erzeugt werden. Die Basisklasse TENTITY besitzt alle notwendigen Funktionen zur Verwaltung ihrer Ports, und um Verbindungen zu anderen Modellkomponenten herzustellen.

Wie man aus Abbildung 2.5 ersehen kann, wird die Modellhierarchie direkt in eine Objekthierarchie umgesetzt. Untergeordnete Modellkomponenten sind dabei Felder ihrer übergeordneten Komponeten. Zusätzlich kennt jede Komponente ihre übergeordnete Komponente, so daß auch eine Kommunikation innerhalb der Modellhierarchie von unten nach oben möglich ist.

Die Kommunikation zwischen verschiedenen Modellkomponenten kann auf drei unterschiedliche Arten erfolgen. Komponenten, die auf der gleichen Hierarchieebene liegen, werden über ihre Ports miteinander verbunden und können so Nachrichten untereinander austauschen. Delegation ist eine zweite Kommunikationsform, die hauptsächlich zwischen Komponenten benachbarter Hierarchieebenen stattfindet. Da jede Komponente untergeordnete Modellelemente als Felder enthält, kennt sie deren Typ und kann die jeweiligen Methoden direkt aufrufen. Die dritte Möglichkeit basiert auf Ereignissen. Näheres dazu wird in Abschnitt 2.3.4 beschrieben.

### 2.3.3 Verbindung von Modellkomponenten

Ports stellen die Schnittstelle zwischen Modellkomponenten dar. Sie ermöglichen eine hohe Flexibilität beim Aufbau eines Modells, indem sie „Plug and Play“ erlauben. Die Bibliothek unterscheidet Eingabe- von Ausgabeports. Sie kann so sicherstellen, daß nicht versehentlich die Ausgänge zweier Komponenten miteinander verbunden werden. Jede Verbindung kann als gerichtete Punkt-zu-Punkt Verbindung zwischen zwei Ports angesehen werden. Jeder Port besitzt einen eindeutigen lokalen Namen und gehört zu einer Modellkomponente. Zusammen mit dem Namen der Komponente ist so eine eindeutige Identifikation jedes Ports innerhalb des Modells möglich. Alle Ports werden von einem globalen Portmanager verwaltet, der u.a. Methoden zum Verbinden von Ports, Umbenennen von Ports und Aliasing von direkt aufeinanderfolgenden Ports über mehrere Hierarchieebenen hinweg besitzt.

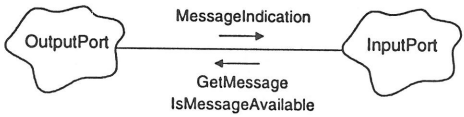



Bild 2.6: Handshake-Protokoll zum Nachrichtenaustausch

Der Nachrichtenaustausch zwischen zwei Ports wird über ein Handshake-Protokoll abgewickelt, das in Abbildung 2.6 in Form eines Objektdiagramms nach Booch [20] dargestellt ist. Das Protokoll besteht aus den drei Meldungen MessageIndication, IsMessageAvailable und GetMessage. Will eine Komponente eine Nachricht versenden, übergibt sie diese an den entsprechenden Ausgabeport zur Übermittlung. Dieser ruft die

MessageIndication-Methode des Empfängerports auf. Es bleibt der empfangenden Komponente überlassen, ob sie die Nachricht sofort oder erst zu einem späteren Zeitpunkt übernehmen will. Soll die Nachricht sofort übernommen werden, ruft der Empfangsport die GetMessage-Methode des Senderports auf. Im zweiten Fall wird der Sender so lange blockiert, bis der Empfänger erneut bei ihm nachfragt, ob eine Nachricht vorhanden ist. Dies geschieht über die IsMessageAvailable-Methode.

Durch dieses einfache Handshake-Protokoll wird der Nachrichtenaustausch zwischen Ports ermöglicht, ohne daß die entsprechenden Modellkomponenten darüber Bescheid wissen müssen, wo und wie eine Nachricht erzeugt oder verarbeitet wird bzw. ob eine Komponente in der Lage ist, die Nachricht zu empfangen. Diese lose Kopplung zwischen Ports ermöglicht es, neue Modellkomponenten sehr einfach zwischen zwei bereits bestehende einzufügen.

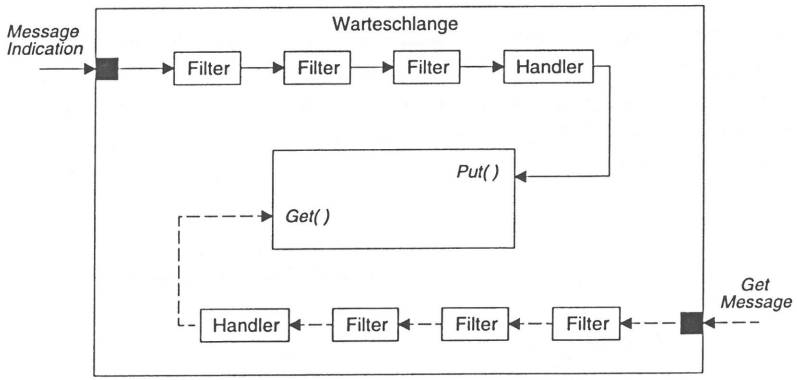



Bild 2.7: Verbindung zwischen Ports und Modellkomponente

Die Verbindung von einem Port mit der Modellkomponente erfolgt über sogenannte Message-Handler. Zwischen einem Port und dem zugehörigen Message-Handler können beliebig viele Filter eingeschoben werden. Auf diese Weise lassen sich sehr einfach Meßklemmen realisieren, über die spezielle Meßgeräte den Nachrichtenaustausch zwischen zwei Komponenten beobachten und so ihre Messungen durchführen können.

Abbildung 2.7 zeigt das Prinzip der Message-Filter und Message-Handler am Beispiel einer Warteschlange. Diese besitzt die internen Methoden Put und Get, um Nachrichten in ihre Datenstruktur ein- bzw. auszutragen. Soll die Warteschlange eine Nachricht empfangen, wird, wie oben beschrieben, die MessageIndication-Methode ihres Eingabeports aufgerufen. Nachdem die Filterkette durchlaufen ist, wird der Aufruf durch den Message-Handler in einen Aufruf der internen Put-Methode umgesetzt. Diese nimmt die Nachricht

entgegen und speichert sie ab. Entsprechend wird ein GetMessage-Aufruf am Ausgabeport durch einen Message-Handler in einen Aufruf der internen Get-Methode umgesetzt.

Die Einführung von Message-Handlern sorgt für eine Entkopplung zwischen Ports und Modellkomponenten. Auf diese Weise ist es möglich, daß Ports universell für verschiedene Modellkomponenten wiederverwendet werden können.

### 2.3.4 Ereignisbearbeitung

Normalerweise werden Ereignisse von einer Modellkomponente zusammen mit ihrem Ereigniszeitpunkt direkt in den Kalender eingetragen. Die Simulationsbibliothek bietet eine Erweiterung in Form einer hierarchischen Ereignisbearbeitung an.

Bei der Ereignisbearbeitung muß zwischen dem Vormerken eines Ereignisses und dessen Ausführung unterschieden werden. Beim Vormerken wird das Ereignis zusammen mit dem Ereigniszeitpunkt an die PostEvent-Methode der Modellkomponente übergeben. Diese kann entscheiden, ob sie das Ereignis selbst bearbeiten will oder nicht. In letzterem Fall übergibt sie es an die ihr übergeordnete Komponente. Gelangt ein Ereignis zum Modell, das auf der obersten Hierarchieebene steht, wird es in den Kalender eingetragen.

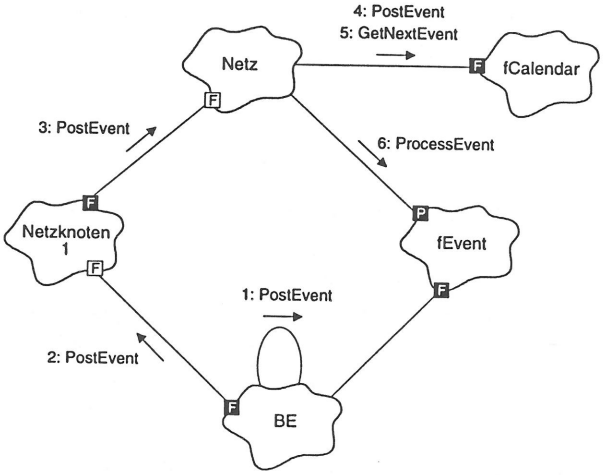



Bild 2.8: Ereignisbearbeitung

Wie bei der ereignisgesteuerten Simulation üblich, trägt die Ablaufsteuerung schrittweise immer das erste Ereignis aus dem Kalender aus und setzt die Systemzeit auf den

jeweiligen Ereigniszeitpunkt. Als nächstes muß das Ereignis bearbeitet, d.h. die Ereignisroutine ausgeführt werden. Dazu wird die ProcessEvent-Methode des Ereignisses aufgerufen. Abbildung 2.8 zeigt die Abläufe beim Vormerken und der Bearbeitung von Ereignissen in Form eines Objektdiagramms.

Die ProcessEvent-Methode, die bei der Bearbeitung eines Ereignisses aufgerufen wird, ist in einer Basisklasse TEVENT definiert. Diese Klasse wird von der Simulationsbibliothek zur Verfügung gestellt und kann vom Anwender dazu benutzt werden, eigene Ereignisse zu definieren. Er muß dazu nur eine neue Klasse ableiten und die ProcessEvent-Methode mit der zugehörigen Ereignisroutine überschreiben.

### 2.3.5 Simulationssteuerung

Die Aufgabe der Simulationssteuerung ist es, den Ablauf des Simulationsprogramms von der Initialisierung bis zur Ausgabe der Ergebnisse durchzuführen. Die beschriebene Bibliothek teilt den Ablauf in einzelne, allgemeine Phasen auf, wie z.B. das Einlesen der Parameter, die Initialisierung des Modells, einen Warmlauf, mehrere Teiltests sowie die Auswertung der Ergebnisse. Sie stellt gleichzeitig sicher, daß die Phasen in der richtigen Reihenfolge durchlaufen werden.

Der Übergang von einer Phase in die nächste ist in der Regel an Bedingungen geknüpft, wie z.B. die Anzahl der Pakete, die bestimmte Stellen des Modells durchlaufen haben. Diese Bedingungen werden durch spezielle Objekte überwacht. Ist eine Abbruchbedingung erfüllt, wird die Simulationssteuerung davon benachrichtigt und kann in die nächste Phase übergehen. Durch Ableiten von speziellen Bibliotheksklassen hat der Anwender die Möglichkeit, sehr leicht eigene Abbruchbedingungen zu definieren.

In einer Simulation gibt es immer mehrere Komponenten, die über die aktuelle Phase Bescheid wissen müssen bzw. über einen Phasenwechsel informiert werden wollen. Dies sind unter anderem alle Statistiken, die nach dem Warmlauf zurückgesetzt und nach jedem Teiltest abgeschlossen werden müssen (siehe dazu auch [42, 111, 112]). Auch in diesem Punkt bietet die Bibliothek dem Anwender eine weitreichende Unterstützung an. Bei der Ablaufsteuerung können Objekte registriert werden, die dann bei einem Phasenwechsel automatisch informiert werden.

# Kapitel 3

# Verkehrslenkungsalgorithmen

Die wenigsten Kommunikationsnetze weisen eine vollständige Vermaschung auf, so daß Sender und Empfänger einer Nachricht direkt miteinander verbunden sind. In den meisten Fällen müssen ein oder mehrere Transferknoten durchlaufen werden. Die Algorithmen, die diesen Weg durch das Netz vom Sender bis zum Empfänger bestimmen, bezeichnet man als *Verkehrslenkungsalgorithmen* (routing algorithms). Aus der Literatur sind mehrere unterschiedliche Verfahren für die Wegeberechnung bekannt. Findet die Berechnung nicht einmalig off-line statt, sind besondere Protokolle notwendig, mit denen die notwendigen Daten über das Netz gesammelt und dem Berechnungsalgorithmus zur Verfügung gestellt werden können. Man spricht in diesem Zusammenhang von einem *Routing-Protokoll*. Die Auswahl eines geeigneten Algorithmus, der zugehörigen Datenstrukturen und gegebenenfalls eines Routing-Protokolls zum Sammeln der Daten ist ein wesentlicher Punkt bei der Konzeption eines Netzes.

Abhängig davon, ob die Kommunikation verbindungsorientiert oder verbindungslos durchgeführt wird, erfolgt die Berechnung eines Wegs einmalig beim Verbindungsaufbau oder für jede einzelne Nachricht (Paket) gesondert. Der Fall, daß ein Weg einmalig beim Verbindungsaufbau berechnet wird, bezeichnet man auch als *session routing*, da der berechnete Weg für die gesamte Dauer einer Kommunikationsbeziehung Gültigkeit besitzt. In der Regel findet die Bestimmung eines Wegs abschnittsweise statt, d.h. ein Knoten berechnet nicht den vollständigen Weg bis zum Ziel, sondern nur den jeweils nächsten Übertragungsabschnitt. Nachdem das Paket über diesen Abschnitt übertragen wurde, bestimmt der empfangende Knoten anhand der Zielinformation im Paketkopf wiederum den nächsten Abschnitt. Dieses abwechselnde Berechnen eines Übertragungsabschnitts und Übertragen über diesen setzt sich solange fort, bis das Paket beim Zielknoten angekommen ist. Mit *source routing* wird ein Verfahren bezeichnet, bei dem der vollständige Weg mit allen Transferknoten bereits beim Sender berechnet und in den Paketkopf eingetragen wird. Ein Transferknoten wertet diese Information aus und überträgt das Paket entlang des vorberechneten Wegs.

Die Anforderungen, die an einen idealen Verkehrslenkungsalgorithmus gestellt werden, sind [14, 145]:

- **Korrektheit**: Der Algorithmus muß einen gültigen Weg liefern;
- **Einfachheit**: Der Algorithmus soll so wenig wie möglich Rechenkapazität im Knoten und Übertragungskapazität durch die Übertragung von Protokollinformation beanspruchen;
- **Robustheit**: Ist ein Netz einmal in Betrieb genommen, hat es eine sehr lange Betriebsdauer, während der die verschiedensten Hardware- und Softwarefehler auftreten können. Der Algorithmus muß robust gegen solche Fehler sein. Das Netz sollte nicht nach jedem Fehler abgeschaltet und von neuem in Betrieb genommen werden müssen;
- **Stabilität**: Der Algorithmus muß gegen eine akzeptable Lösung konvergieren. Er darf nicht zwischen zwei oder mehreren Lösungen oszillieren, wenn er sich an veränderte Last- oder Topologieverhältnisse anpaßt;
- **Fairness**: Der Algorithmus muß alle Verbindungswünsche bzw. Pakete unter Berücksichtigung der vorgegebenen Prioritäten gleich behandeln;
- **Optimalität**: Der Algorithmus soll den „besten" Weg liefern, der die mittlere Paketverzögerung minimiert und den Durchsatz maximiert.

Einige dieser Anforderungen sind nicht unbedingt immer gleichzeitig zu erfüllen. Dazu gehören z.B. Fairness, Minimierung der Paketverzögerung und Maximierung des netzweiten Durchsatzes. Je nach geplantem Einsatzbereich müssen hier u.U. Prioritäten gesetzt werden.

## 3.1 Allgemeines Klassifizierungsschema

Aus der Literatur sind zahlreiche Verkehrslenkungsalgorithmen bekannt. Für die Klassifizierung dieser Algorithmen sind mehrere, zum Teil unterschiedliche Ansätze bekannt (z.B. [14, 145], [78] oder [126]). In Abbildung 3.1 sind die Zusammenhänge zwischen verschiedenen Klassen von Verkehrslenkungsalgorithmen entsprechend [14, 145] dargestellt.

Die Verfahren lassen sich in zwei Hauptgruppen einteilen. *Nicht-adaptive* Algorithmen berücksichtigen bei der Wegewahl keine aktuellen Daten oder Abschätzungen des momentanen Verkehrs oder der aktuellen Topologie. Die Wege werden bereits im voraus off-line berechnet, in Tabellen abgespeichert und anschließend in die jeweiligen Netzknoten geladen. Die *adaptiven* Verfahren versuchen, die aktuelle Topologie des Netzes oder die momentanen Lastverhältnisse in die Berechnungen mit einzubeziehen. Die Gruppe

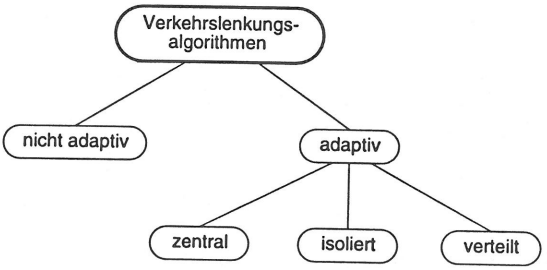



Bild 3.1: Klassifizierungsschema für Verkehrslenkungsalgorithmen

der adaptiven Verkehrslenkungsverfahren läßt sich ihrerseits in drei Untergruppen unterteilen. Das Kriterium für die Unterteilung ist der Ort, an dem der jeweilige Algorithmus abläuft, und, welche Daten den Berechnungen zugrunde liegen.

Bei der *zentralen Verkehrslenkung* (centralized routing) wird die Berechnung der Wege von einem zentralen Knoten, dem sogenannten *routing control center* (RCC), durchgeführt. Alle notwendigen Daten über den Zustand bzw. die Belastung der Knoten oder Leitungen werden von den Netzknoten in regelmäßigen Abständen an das Kontrollzentrum übermittelt und dort gespeichert. Auf Grund der gesammelten Informationen berechnet das RCC für jeden einzelnen Knoten eine Verkehrslenkungstabelle, in der für jeden Zielknoten der berechnete Weg bzw. der nächste Übertragungsabschnitt eingetragen ist. Diese Tabellen werden anschließend an die jeweiligen Knoten gesendet.

Ein Nachteil der zentralen Verkehrslenkung ist, daß sie sehr stark vom ordnungsgemäßen Betrieb des Kontrollzentrums abhängt. Fällt es aus, sind die Folgen meist katastrophal. Die Knoten können zwar mit der zuletzt empfangenen Tabelle weiterarbeiten, die Informationen sind aber sehr schnell nicht mehr aktuell. Aus diesem Grund gibt es in Netzen mit zentraler Verkehrslenkung meistens mehrere Knoten, die die Funktion des Kontrollzentrums im Bedarfsfall übernehmen können oder sogar parallel arbeiten. Ein weiterer Nachteil der zentralen Verkehrslenkung besteht darin, daß die Leitungen um das Kontrollzentrum durch das Sammeln der Daten und das Versenden der Tabellen stark belastet werden.

Bei der *isolierten Verkehrslenkung* (isolated routing) berechnet jeder Netzknoten die Wege zu allen anderen Knoten selbständig. Er verwendet dazu nur lokale Informationen. Ein direkter Austausch von Daten zwischen den Knoten findet nicht statt.

Die dritte Gruppe bilden die *verteilten Verkehrslenkungsverfahren* (distributed routing). Bei dieser Klasse tauschen die einzelnen Knoten mit Hilfe eines speziellen Protokolls

Informationen über den Zustand ihrer Leitungen bzw. ihre Belastung untereinander aus. Basierend auf diesen Daten bestimmt jeder Knoten die Wege selbständig.

Eine weitere Klassifikationsmöglichkeit wurde in [126] vorgeschlagen. Die Algorithmen können danach unterschieden werden, ob sie versuchen, den jeweils besten Weg für einzelne Quelle-Ziel-Paare zu berechnen, oder ob eine netzweite Optimierung z.B. der Verzögerungszeiten aller Quelle-Ziel-Paare vorgenommen wird. Die erst genannte Klasse ist die Klasse der *Shortest Path-Algorithmen*. Hier wird jedem Übertragungsabschnitt eine bestimmte „Länge", basierend auf einer frei definierbaren Metrik, zugewiesen. Die Länge des gesamten Wegs ergibt sich aus der Summe der Längen der einzelnen Abschnitte. Ein Shortest Path-Algorithmus berechnet für jedes Quelle-Ziel-Paar den Weg, für den die oben definierte Länge am kürzesten ist.

Bei der *optimalen Verkehrslenkung* (optimal routing) werden einzelne Quelle-Ziel-Paare nicht getrennt voneinander betrachtet, sondern es wird versucht, eine Optimierung für das gesamte Netz vorzunehmen. Grundlage für die Optimierung bildet ein Flußmodell [18], bei dem der Verkehrsfluß für jedes Quelle-Ziel-Paar bekannt sein muß. Mit Hilfe verkehrstheoretischer Ansätze werden aus diesen Verkehrsflüssen die Verzögerungszeiten abgeleitet, deren Summe über das ganz Netz hinweg minimal werden muß. Dabei kann es durchaus sein, daß einzelne Quelle-Ziel-Paare nicht den Weg mit der geringsten Verzögerungszeit zugewiesen bekommen, wenn dadurch die Gesamtverzögerung reduziert werden kann. Näheres dazu ist in Abschnitt 3.3.4 oder in [48, 123] zu finden.

Die zweite Klassifikation widerspricht der zuerst besprochenen nicht, sondern ergänzt diese vielmehr. Shortest Path-Algorithmen können sowohl für die zentrale als auch für die verteilte Verkehrslenkung eingesetzt werden. Ferner ist es denkbar, die off-line-Berechnung der Verkehrslenkungstabellen bei der nicht-adaptiven Verkehrslenkung mit Hilfe eines Shortest Path-Algorithmus durchzuführen. Optimale Verkehrslenkungsalgorithmen werden in realen Nezten bisher nicht direkt eingesetzt [126]. Sie finden hauptsächlich bei der Topologieplanung bzw. bei der Erstellung der Verkehrslenkungstabellen für die nichtadaptive Verkehrslenkung Verwendung.

Schließlich kann noch zwischen *hierarchischer* und *nicht-hierarchischer Verkehrslenkung* unterschieden werden. Bei der hierarchischen Verkehrslenkung [32, 86] wird das Netz in mehrere, hierarchisch angeordnete *Domänen* (domains) unterteilt. Innerhalb einer Domäne findet die Verkehrslenkung nach einem der oben beschriebenen Verfahren statt. Ist der Zielknoten innerhalb einer Domäne nicht bekannt, wird ein Paket automatisch eine Hierarchieebene nach oben gereicht. Bei der nicht-hierarchischen Verkehrslenkung unterbleibt diese Aufteilung, und das gesamte Netz wird als eine einzige große Domäne betrachtet.

## 3.2 Algorithmen für die Durchschaltevermittlung

In diesem Kapitel sollen kurz die wichtigsten Verfahren für durchschaltevermittelnde Netze vorgestellt und die wesentlichen Unterschiede gegenüber den Verfahren bei paketvermittelnden Netzen aufgezeigt werden. Eine detailliertere Übersicht über bestehende Verfahren und weiterführende Literatur sind in [29, 156] enthalten.

Die vorgestellten Verfahren werden hauptsächlich in internationalen Telefonnetzen eingesetzt. Bei der Durchschaltevermittlung kann eine Verbindung nur dann angenommen werden, wenn auf dem ganzen Weg vom Sender bis zum Empfänger ein freier Kanal vorhanden ist, der durchgeschaltet werden kann. Im Gegensatz dazu werden bei der Paketvermittlungstechnik alle Pakete in den Knoten zwischengespeichert. Deshalb können kurzzeitig mehr Pakete ankommen als Übertragungskapazität zur Verfügung steht. Dies resultiert lediglich in größeren Warte- bzw. Transferzeiten oder vereinzelt in Paketverlusten. Die Sicherungsmechanismen der höheren Schichten erkennen den Verlust eines Pakets und wiederholen die Übertragung. Dieser Unterschied muß von den Verkehrslenkungsalgorithmen berücksichtigt werden. Als Kriterium für die Güte eines Wegs wird deshalb bei der Durchschaltevermittlung meistens die Blockierwahrscheinlichkeit herangezogen. Bei der Paketvermittlung dient dagegen die Transferzeit oft als Maß für die Bewertung eines Wegs. Ein weiterer Unterschied für die Algorithmen besteht in der Topologie der Netze, für die sie Wege berechnen sollen. Telefonnetze sind in der Regel hierarchisch aufgebaut und weisen eine starke Vermaschung auf. Dies hat zur Folge, daß die Wege sehr kurz sind, was sich vereinfachend auf die Berechnung auswirkt. Oftmals wird die Anzahl der erlaubten Übertragungsabschnitte auf zwei begrenzt (siehe z.B. [7, 118]). Paketvermittelnde Datennetze sind häufig weniger strukturiert und die Wege beinhalten mehrere Transferknoten (hops).

Eine früher häufig eingesetzte Technik war die *alternative Verkehrslenkung* (alternate routing) in Verbindung mit dem Überlaufprinzip [29]. Jeder Vermittlungsknoten kannte zu jedem Ziel mehrere geordnete Alternativwege. Konnte eine Verbindung über den Erstweg nicht durchgeschaltet werden, weil kein freier Kanal mehr vorhanden war, wurde der Zweitweg, gegebenenfalls der Drittweg etc. verwendet. Nach der Einführung der Rechnersteuerung wurden auch bei der Verkehrslenkung *dynamische* Verfahren (dynamic routing) entwickelt [5, 6, 13]. Bei der dynamischen Verkehrslenkung kann man zwei Klassen unterscheiden: das *zeitabhängige dynamische Verkehrslenkung* (time-dependent dynamic routing) und die *zustandsabhängige dynamische Verkehrslenkung* (state-dependent dynamic routing).

Verfahren der ersten Klasse berücksichtigen, daß der Verkehr im Netz zu verschiedenen Tageszeiten unterschiedlich ist. Ein Tag wird in mehrere Intervalle aufgeteilt, für die

Wege auf Grund von Verkehrsschätzungen im voraus berechnet und in Tabellen abgelegt werden. Zu den entsprechenden Tageszeiten wird dann zwischen den vorberechneten Tabellen umgeschaltet [7].

Die zustandsabhängigen Verfahren berücksichtigen bei der Berechnung der Wege die momentane Belegung der Leitungsbündel. Als Auswahlkriterium kann dabei z.B. die Zahl der freien Leitungen eines Bündels [8, 54] oder die Blockierwahrscheinlichkeit [91, 118] herangezogen werden. In [83] wird ein lernender Automat zur Auswahl der Wege benutzt. Beispiele aus dieser Klasse findet man z.B. in [8, 26, 54, 90, 143, 160].

## 3.3 Algorithmen für die Paketvermittlung

### 3.3.1 Nicht-adaptive Verkehrslenkung

Das einfachste Verkehrslenkungsverfahren ist die *statische Verkehrslenkung* (static routing, fixed directory routing). Jeder Knoten besitzt eine Tabelle, die sogenannte Verkehrslenkungstabelle, mit jeweils einem Eintrag für jeden Zielknoten innerhalb des Netzes. Dieser Eintrag bestimmt den nächsten Nachbarknoten, zu dem die Pakete mit dem entsprechenden Ziel übermittelt werden. Die Tabelle wird off-line berechnet und bleibt während des Betriebs unverändert. Gibt es mehr als einen Weg zu einem Zielknoten, kann das *multipath routing* (auch bifurcated routing oder split traffic routing) eingesetzt werden. Die Verkehrslenkungstabelle besitzt hier mehrere gewichtete Einträge pro Zielknoten. Die Gewichtung kann als Wahrscheinlichkeit interpretiert werden, mit der die Einträge ausgewählt werden [11, 21, 126, 145].

Mit *flooding* wird eine Methode bezeichnet, bei der jedes Paket auf allen zur Verfügung stehenden Leitungen, ausgenommen der Empfangsleitung, weitergesendet wird. Da bei diesem Verfahren theoretisch unendlich viele Pakete erzeugt werden, müssen Mechanismen vorgesehen werden, um die Zahl der Pakete zu begrenzen. Dies kann über einen Zähler im Paketkopf geschehen, der die Anzahl der Transferknoten beschränkt. Jeder Knoten, der ein Paket empfängt, dekrementiert den Zähler im Paketkopf. Erreicht der Zählerstand Null, wird das Paket vernichtet. Eine zweite Methode, die Zahl der Pakete zu begrenzen, besteht darin, den Inhalt eines Pakets eine gewisse Zeit abzuspeichern. Ein Paket wird nur dann weitergesendet, wenn sein Inhalt noch nicht bekannt ist.

Beim *selective flooding* werden für jeden Knoten die Leitungen festgelegt, auf denen er Pakete weitersendet. Auf diese Weise kann die Anzahl der Pakete nochmals reduziert werden. Da dadurch aber der Lawineneffekt nicht vollständig unterbunden werden kann, ist zusätzlich eine der oben beschriebenen Maßnahmen erforderlich.

Wegen der großen Zahl an erzeugten Paketen und der damit verbundenen Gefahr der Überlastung des Netzes, wird flooding nur in Ausnahmefällen auf Datenpakete angewendet. Es ist dagegen ein häufig eingesetztes Verfahren für Broadcast-Meldungen oder die Verbreitung von Topologiedaten im Zusammenhang mit Shortest Path-Algorithmen (vgl. Abschnitt 3.3.2).

### 3.3.2 Shortest Path Routing

Ein sehr weitverbreitetes Verfahren, das sowohl für die zentrale [117, 150] als auch für die verteilte [18, 102, 145] Verkehrslenkung eingesetzt werden kann, ist das Shortest Path Routing. Basierend auf einer beliebigen Metrik wird jedem Übertragungsabschnitt eine bestimmte „Länge“ zugeordnet. Die Länge eines Wegs vom Quell- bis zum Zielknoten ergibt sich aus der Summe der Längen der jeweiligen Teilabschnitte. Durch spezielle Algorithmen wird der jeweils kürzeste Weg zu einem Zielknoten berechnet und alle Pakete über diesen Weg geleitet.

Der Wahl der Metrik kommt bei diesen Verfahren eine große Bedeutung zu. In der Praxis wird jedem Abschnitt häufig die Länge „1“ zugeordnet [43, 52, 61, 110]. Die Gesamtlänge eines Wegs ergibt sich demnach aus der Anzahl der Knoten, die ein Paket durchlaufen muß (sog. hop count). Man geht hier davon aus, daß ein Paket um so länger unterwegs ist, je mehr Knoten es durchlaufen muß. Speziell in heterogenen Netzen mit Leitungen unterschiedlicher Übertragungskapazität und Knoten unterschiedlicher Leistungsfähigkeit ist dies eine sehr grobe Näherung. Die Kapazität einer Leitung [125, 126, 142] oder die Anzahl der wartenden Pakete in den jeweiligen Ausgangspuffern [28] können ebenfalls als Maß für die Länge eines Abschnitts dienen. In manchen Systemen wird versucht, die Transferzeit zwischen zwei Knoten zu messen [104, 119]. Mit der Wahl einer geeigneten Metrik befassen sich außerdem [31, 58, 84].

Für die Berechnung der kürzesten Wege bei bekannter Topologie des Netzes sind aus der Literatur mehrere Verfahren bekannt [2, 18, 126]. Die bekanntesten Verfahren sind der Algorithmus nach Dijkstra [40], sowie Variationen der Arbeiten von Floyd [44] und Ford [45]. Alle diese Algorithmen berechnen die Wege iterativ, wobei bei Dijkstra über die Weglänge, bei Floyd über die Transferknoten und bei Ford über die Kanten des Topologiegraphen iteriert wird. Die Algorithmen unterscheiden sich z.T. auch im Ergebnis, das man am Ende der Iteration erhält. Floyds Algorithmus berechnet die kürzesten Wege zwischen allen Quelle-Ziel-Paaren auf einmal, während Dijkstras Algorithmus nur die Wege von einem Quellknoten zu allen Zielknoten berechnet. Um das gleiche Ergebnis wie bei Floyds Algorithmus zu erhalten, muß Dijkstras Algorithmus mehrfach angewendet werden.

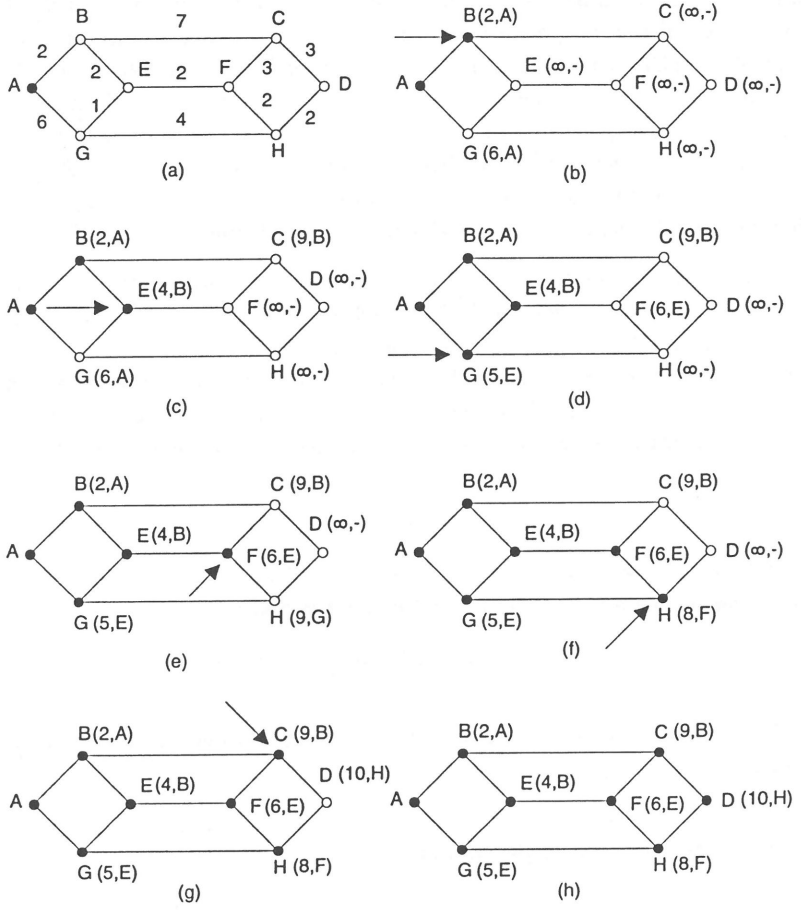



Bild 3.2: Beispiel für den Dijkstra-Algorithmus

Der Ablauf des Dijkstra-Algorithmus soll anhand des Beispiels in Abbildung 3.2 kurz erläutert werden. Jedem Knoten wird eine Markierung zugeordnet, die aus seiner momentanen Entfernung zum Startknoten und dem Vorgänger entlang des betrachteten Wegs besteht. Es gibt temporäre und permanente Markierungen. Während der einzelnen Iterationsschritte können sich temporäre Markierungen verändern. Hat man den kürzesten Weg zu einem Knoten gefunden, wird dessen Markierung permanent. Knoten mit einer permanenten Markierung sind in Abbildung 3.2 durch einen dunklen Punkt gekennzeichnet.

Zu Beginn des Algorithmus werden alle Knoten, die nicht unmittelbar mit dem Startknoten verbunden sind, mit der Entfernung $\infty$ initialisiert. Direkte Nachbarn des Startknotens erhalten eine temporäre Markierung entsprechend ihrer Entfernung. Nur der Startknoten erhält eine permanente Markierung (Abbildung 3.2a). Bei jedem Iterationsschritt erhält derjenige Knoten eine permanente Markierung, dessen Entfernung zum Startknoten am geringsten ist (Knoten B in Abbildung 3.2b). Anschließend werden die temporären Markierungen seiner Nachbarn neu berechnet (Abbildung 3.2c). Der nächste Iterationsschritt beginnt und Knoten E erhält eine permanente Markierung. Der Algorithmus ist beendet, wenn alle Knoten eine permanente Markierung besitzen.

Ein zweiter, wichtiger Shortest Path-Algorithmus, der in verschiedenen Variationen [9, 10, 30, 68, 102, 103] für die verteilte Verkehrslenkung eingesetzt wird, ist der Bellman-Ford-Algorithmus [18]. Mit $d_{jk}$ werde die Länge der Leitung zwischen Knoten $j$ und $k$ und mit $D_{ik}^{(h)}$ die Länge des kürzesten Wegs vom Knoten $i$ zu Knoten $k$ bezeichnet, wobei dieser maximal $h$ Leitungen (hops) enthalten darf. Der einfacheren Darstellung wegen gelte $d_{jk} = \infty$ falls keine Leitung zwischen den Knoten $j$ und $k$ existiert. Ferner gilt:

$$D_{ii}^{(h)} = 0 \qquad \forall\, h. \tag{3.1}$$

Setzt man $D_{ik}^{(0)} = \infty$ für alle $k \neq i$, so lassen sich die kürzesten Wege von $i$ zu allen anderen Knoten iterativ nach folgender Gleichung berechnen:

$$D_{ik}^{(h+1)} = \min_j \left[ D_{ij}^{(h)} + d_{jk} \right] \qquad \forall\, k \neq i \tag{3.2}$$

Die Iteration ist beendet, wenn $h = N - 1$ ist, wobei $N$ die Anzahl der Knoten im Netz angibt.

### 3.3.3 Adaptive Verkehrslenkung

In Abschnitt 3.1 wurden die adaptiven Verkehrslenkungsverfahren in drei Untergruppen gegliedert. Bei der zentralen Verkehrslenkung sammelt eine zentrale Instanz Daten über den Status und die Belastung der Leitungen und berechnet daraus die Verkehrslenkungstabellen. Diese werden anschließend an die jeweiligen Knoten verteilt. Im Tymnet z.B., einem kommerziellen Paketvermittlungsnetz, wird diese zentrale Verkehrslenkung eingesetzt [117, 125, 126, 150]. Allerdings werden hier die berechneten Tabellen nicht an die Knoten verteilt, sondern das RCC erzeugt für jeden Verbindungswunsch ein „Nadelpaket", das den Weg vom Quell- zum Zielknoten enthält.

Bei der isolierten Verkehrslenkung findet die Berechnung der Wege dezentral in jedem Knoten statt. Grundlage für den Berechnungsalgorithmus sind nur lokale Informationen.

Ein sehr einfacher Vertreter dieser Gruppe ist der sogenannte *hot-potato* Algorithmus [12]. Bei ihm werden Pakete immer über den Abschnitt übertragen, der die geringste Anzahl an wartenden Paketen im zugehörigen Ausgangspuffer besitzt.

Der *backward-learning* Algorithmus [145] basiert auf der Annahme, daß die Entfernung zwischen zwei Knoten in beiden Richtungen identisch ist. Jedes Paket führt einen speziellen Zähler im Paketkopf mit, der im Ursprungsknoten zurückgesetzt und in jedem Transferknoten erhöht wird. Durch einfaches Beobachten der vorbeikommenden Pakete lernt jeder Knoten die Entfernungen zu anderen.

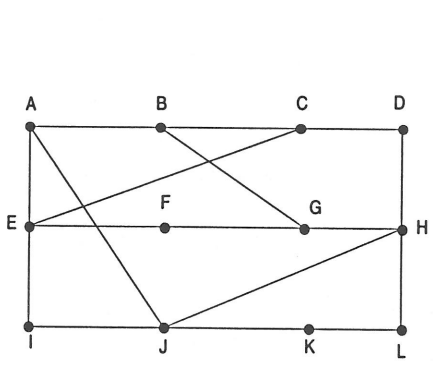
Das *delta routing* [120] ist eine Mischform aus zentraler und isolierter Verkehrslenkung. Jeder Knoten sendet Informationen über die Auslastung der Leitungen zu einem zentralen Kontrollzentrum. Anhand dieser Daten werden dort Verkehrslenkungstabellen berechnet, die mehrere Alternativen enthalten. Die Auswahl eines bestimmten Wegs erfolgt dezentral in jedem Knoten unter Berücksichtigung der momentanen Lastsituation. Die Anzahl der Alternativen, die das zentrale Kontrollzentrum berechnet, wird durch einen Faktor $\delta$ bestimmt. Wird $\delta = 0$ gesetzt, berechnet das Kontrollzentrum nur einen Weg und man erhält die zentrale Verkehrslenkung. Für sehr hohe Werte von $\delta$ erhält man ein nahezu dezentrales Verfahren.

Bei der dritten Gruppe der adaptiven Verkehrslenkungsverfahren entsprechend Abbildung 3.1, der verteilten Verkehrslenkung, tauschen alle Knoten untereinander Informationen aus. Es gibt zwei vollkommen unterschiedliche Realisierungsformen: die sogenannten *Link-State-Protokolle* und die *Distance-Vector-Protokolle*. Link-State-Protokolle arbeiten mit einem Dijkstra-Algorithmus, der dezentral in jedem Knoten abläuft. Jeder Knoten versendet die Informationen über die Länge der Leitungen zu seinen direkten Nachbarn mittels Broadcast an alle anderen Knoten. Diese Informationen werden von den einzelnen Knoten gesammelt und dienen als Berechnungsgrundlage für den Shortest Path-Algorithmus.

Für ein korrektes Arbeiten dieser Link-State-Algorithmen ist ein schnelles und sicheres Verfahren zum Austausch (Broadcast) der Topologiedaten (Leitungslängen) von entscheidender Bedeutung. In [19, 67, 146] wird die Verteilung der Topologieinformationen entlang des sogenannten *spanning trees* vorgeschlagen. Unter einem spanning tree versteht man einen Baum, dessen Wurzel der sendende Knoten ist und der jeden Knoten genau einmal enthält. Bei der Verteilung der Topologiedaten entlang eines spanning trees wird das Netz am wenigsten belastet. Es ergeben sich aber Probleme, wenn sich die Topologie des Netzes ändert solange noch Broadcast-Meldungen unterwegs sind [113, 140]. Fällt eine Leitung aus, die Teil des Baums ist, erreicht eine Meldung unter Umständen nicht alle Knoten. Aus diesem Grund wird häufig flooding zum Verteilen der Topologieinformationen eingesetzt [113, 119].

Werden die Topologiedaten regelmäßig verschickt, ist eine Folgenummernsteuerung notwendig, um zu verhindern, daß Knoten mit veralteten Informationen arbeiten. In [1, 41, 131, 141] wurden Protokolle vorgeschlagen, die ohne Folgenummern auskommen. Diese sind aber entweder sehr laufzeitineffizient oder arbeiten nicht unter allen Umständen korrekt [140].

Distance-Vector-Protokolle basieren auf einer verteilten Version des Bellman-Ford-Algorithmus. Die Iteration aus Gleichung 3.2 wird verteilt durchgeführt. Während jedes Iterationsschritts sendet ein Knoten einen Vektor, in dem seine Entfernung zu allen anderen Knoten des Netzes eingetragen ist, an alle Nachbarn. Gleichzeitig empfängt er von diesen entsprechende Vektoren. Da jeder Knoten zusätzlich die Entfernung zu seinen Nachbarknoten kennt, kann er die Minimierung entsprechend Gleichung 3.2 durchführen. Ändern sich die Entfernungsvektoren während eines Iterationsschritts, verschickt er die neuen Vektoren erneut an alle Nachbarn. Damit beginnt der nächste Iterationsschritt. In Abbildung 3.3 ist ein Iterationsschritt aus Sicht des Knoten $j$ dargestellt. Ein Beweis für die Konvergenz der verteilten Iteration ist in [16] enthalten.




Distanzvektor von Knoten:

| Ziel | A | I | H | K | Entf. | Kn. |
|---|---|---|---|---|---|---|
| A | 0 | 24 | 20 | 21 | 8 | A |
| B | 12 | 26 | 31 | 28 | 20 | A |
| C | 25 | 18 | 19 | 36 | 28 | I |
| D | 40 | 27 | 8 | 24 | 20 | H |
| E | 14 | 7 | 30 | 22 | 17 | I |
| F | 23 | 20 | 19 | 40 | 30 | I |
| G | 18 | 31 | 6 | 31 | 18 | H |
| H | 17 | 20 | 0 | 19 | 12 | H |
| I | 21 | 0 | 14 | 22 | 10 | I |
| J | 9 | 11 | 7 | 10 | 0 | - |
| K | 24 | 22 | 22 | 0 | 6 | K |
| L | 29 | 33 | 9 | 9 | 15 | K |

Entfernung: JA=8 JI=10 JH=12 JK=6 neue Tabelle Knoten J

Bild 3.3: Beispiel für ein Distance-Vector-Protokoll

Der bekannteste Vertreter aus der Klasse der Distance-Vector-Protokolle ist das Verfahren, das ursprünglich im ARPANET eingesetzt wurde [18, 101, 102, 103]. Dieses Netz wurde zu Beginn der 70er Jahre mit Unterstützung des amerikanischen Verteidigungsministeriums als Forschungsnetz entwickelt und in Betrieb genommen [115]. Im Laufe der Jahre wurde es in mehrere Teile aufgespalten und ist heute die oberste Ebene des Internet, des weltweit größten Datennetzes. Die Verkehrslenkung im ARPANET wurde nach einiger Zeit auf ein Link-State-Protokoll umgestellt [104].

Ein ausführlicher Vergleich der Eigenschaften von Link-State- und Distance-Vector-Protokollen ist in [133, 134] zu finden. Die durchgeführten Untersuchungen basieren auf Simulationsstudien.

Wird eine Verbindung abschnittsweise aufgebaut bzw. ein Paket abschnittsweise vermittelt, d.h. wird kein source routing verwendet, kann es auf Grund der verteilten Berechnung sehr leicht zu Schleifen in den Wegen kommen. Diese können bei Link-State-Protokollen entstehen, wenn die Knoten von unterschiedlichen Topologiedaten ausgehen. Bei Distance-Vector-Protokollen können Schleifen vorübergehend während der Iteration entstehen. Werden keine besonderen Maßnahmen zur Erkennung und Behebung von Schleifen getroffen, können Pakete beliebig lange im Kreis laufen und das Netz stark belasten.

In Abbildung 3.4 ist ein einfaches Beispiel für die Schleifenbildung bei einem Distance-Vector-Algorithmus dargestellt. Die Entfernung zwischen den Knoten A, B und C betrage jeweils eine Einheit. Nach dem Ausfall der Leitung zwischen den Knoten B und C berechnet B einen neuen Entfernungsvektor und sendet diesen an A. Als Grundlage verwendet er dabei den Vektor, den er von A unmittelbar vor dem Ausfall empfangen hat. In der Abbildung sind für jeden Iterationsschritt die Entfernungen zum Knoten C eingezeichnet. Obwohl dieser nach dem Ausfall nicht mehr erreichbar ist, dauert es einige Zeit, bis die anderen Knoten die korrekte Entfernung zu C berechnet haben. Das Problem ist in der Literatur auch unter dem Namen *counting to infinity* bekannt. Ein Paket, das zum Knoten C gesendet werden soll, bevor die Iteration beendet ist, würde ständig zwischen den Knoten A und B hin- und herpendeln.

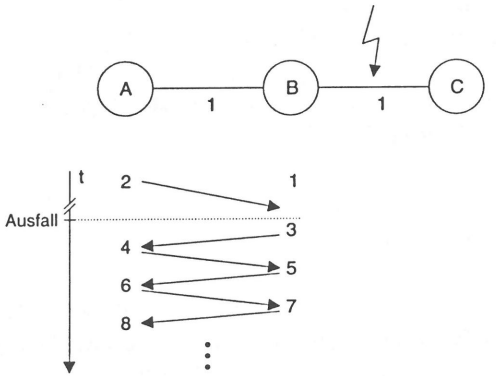



Bild 3.4: Beispiel für das „counting to infinity" Problem

Während Schleifen, die nur aus zwei Knoten bestehen, noch sehr einfach zu beheben sind, ist es wesentlich schwieriger, Schleifen aus mehreren Knoten zu vermeiden. In [51, 52,

53, 81] werden verschiedene, sehr aufwendige Protokolle vorgeschlagen, die den Austausch der Entfernungsvektoren genauer regeln. In [116, 136] wird versucht das Problem dadurch zu lösen, daß in den Distanzvektoren neben den Entfernungen zusätzliche Informationen enthalten sind. [116] kann die Bildung von temporären Schleifen trotzdem nicht vollständig ausschließen. In [136] sind für die Bestimmung der Zusatzinformationen globale Daten über das Netz notwendig. Dies erschwert eine vollständig verteilte Lösung insbesondere dann, wenn diese globalen Daten Änderungen unterworfen sein können.

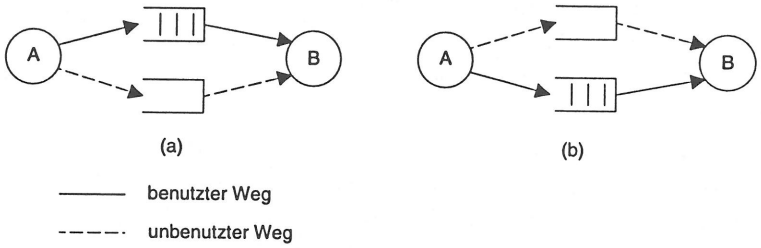



Bild 3.5: Beispiel für die Dynamik von adaptiven Algorithmen

Wird bei den adaptiven Verfahren die momentane Lastsituation mitberücksichtigt, können dadurch Stabilitätsprobleme auftreten. Dies läßt sich sehr einfach anhand von Abbildung 3.5 veranschaulichen. In Abbildung 3.5a werden alle Pakete über den oberen Weg geleitet. Dies führt dazu, daß dieser entsprechend stark belastet ist und folglich viele Pakete in seiner Ausgangswarteschlange warten. Die Warteschlange des unteren Weges ist leer. Erfolgt nun eine Neuberechnung der kürzesten Wege, ist der untere Weg kürzer, d.h. alle Pakete werden von nun ab über den unteren Weg gelenkt. Nach kurzer Zeit ist jetzt die untere Warteschlange voll und die obere leer (Abbildung 3.5b). Bei der nächsten Neuberechnung drehen sich die Verhältnisse entsprechend wieder um. Eine genauere Betrachtung dieser Probleme erfolgt in [155, 163].

### 3.3.4 Optimale Verkehrslenkung

Shortest Path-Algorithmen lenken jedes Paket bzw. jede Verbindung über den jeweils kürzesten Weg. Bei der Berechnung der kürzesten Wege wird die gegenseitige Beeinflussung der Verbindungen nicht unmittelbar berücksichtigt. Bei adaptiven Verfahren wird erst im darauffolgenden Iterationsschritt das Ergebnis des vorangegangenen Schritts beurteilt und u.U. entsprechend korrigiert. Dies führt zu dem oben aufgezeigten Schwingverhalten [139]. Eine ideale Lösung für obiges Beispiel würde den Verkehr gleichmäßig auf beide Wege aufteilen.

Daß der kürzeste Weg nicht immer der beste ist, läßt sich auch anhand des Paradoxons von Braess [22, 34, 82] belegen. Die Bewertung der Wege erfolgt mit Hilfe einer Kostenfunktion, die für jede Kante definiert wird und vom Verkehr über diese Kante abhängt (Im Zusammenhang mit optimaler Verkehrslenkung spricht man von den Kosten einer Kante und nicht von deren Länge). Für das Netz in Abbildung 3.6 seien die Kostenfunktionen $D_{ij}$ der Kante $(i, j)$ in Abhängigkeit des Verkehrsflusses $F_{ij}$ wie folgt definiert:

$$\begin{aligned} D_{AB}(F_{AB}) &= 10F_{AB} \\ D_{AC}(F_{AC}) &= 50 + F_{AC} \\ D_{BC}(F_{BC}) &= 10 + F_{BC} \\ D_{BD}(F_{BD}) &= 50 + F_{BD} \\ D_{CD}(F_{CD}) &= 10F_{CD} \end{aligned} \tag{3.3}$$

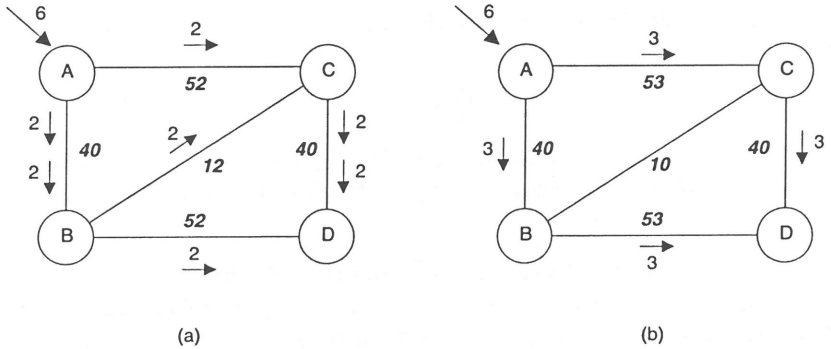



Bild 3.6: Das Paradoxon von Braess

Es soll ein Fluß von 6 Paketen von Knoten A zu Knoten D transportiert werden. In Abbildung 3.6a wird dieser gleichmäßig auf die Wege (A, B, D), (A, C, D) und (A, B, C, D) aufgeteilt. Die resultierenden Kosten für jede Kante sind in der Abbildung fett gedruckt dargestellt. Jedes Paket verursacht höhere Kosten, wenn es auf einen anderen Weg umgelegt wird. Dieser Zustand ergibt sich demnach, wenn jedes Paket für sich über den kürzesten Weg gelenkt wird. Die Gesamtkosten für das Netz ergeben sich aus der Summe der Kantenkosten zu $D = 196$. In Abbildung 3.6b ist die optimale Lösung bezüglich der Gesamtkosten dargestellt. Sie reduzieren sich in diesem Fall auf $D = 176$.

Optimale Verkehrslenkungsalgorithmen basieren auf einem Flußmodell entsprechend Abbildung 3.7. Die Verkehrslenkungsaufgabe besteht darin, den Eingangsfluß $r_{AD}$ vom Quellknoten A zum Zielknoten D so zu übertragen, daß die Gesamtkosten $D$ minimal

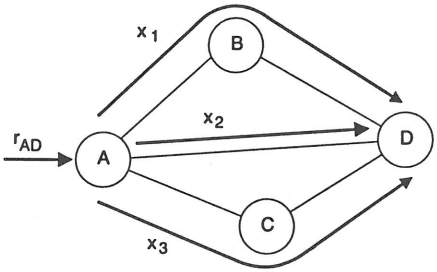



Bild 3.7: Flußmodell für die optimale Verkehrslenkung

werden. Im obigen Beispiel stehen dafür drei Wege zur Verfügung. Für die Flüsse $x_1$, $x_2$ und $x_3$ über diese Wege gilt:

$$x_1 + x_2 + x_3 = r_{AD}. \tag{3.4}$$

Allgemein lautet der Satz über die Flußerhaltung:

$$\sum_{p \in P_w} x_p = r_w. \tag{3.5}$$

Dabei bezeichnet $x_p$ den Fluß entlang des Weges $p$ und $P_w$ die Menge der Wege, die das Quelle-Ziel-Paar $w$ verbinden. Der Weg $p$ selbst besteht aus der Aneinanderreihung von Kanten von der Quelle zum Ziel.

Die Gesamtkosten des Netzes $D$ ergeben sich aus der Summe der Kosten $D_{ij}$ für alle Kanten $(i,j)$. Die Kosten für eine Kante sind vom Fluß $F_{ij}$ über diese Kante abhängig. Dieser berechnet sich aus der Summe der Wegflüsse $x_p$, die über diese Kante gelenkt werden, zu:

$$F_{ij} = \sum_{p \mid (i,j) \in p} x_p. \tag{3.6}$$

Faßt man alle Wegflüsse $x_p$ aller Quelle-Ziel-Paare $w$ zu einem Vektor $\vec{x}$ zusammen, lassen sich die Gesamtkosten wie folgt darstellen:

$$D(\vec{x}) = \sum_{(i,j)} D_{ij}\Big( \sum_{p \mid (i,j) \in p} x_p \Big). \tag{3.7}$$

Das Optimierungsproblem besteht nun darin, Gleichung 3.7 unter Berücksichtigung der Randbedingungen:

$$\begin{aligned} \sum_{p \in P_w} x_p &= r_w \quad \forall\, w \\ x_p &\geq 0 \quad \forall\, p \end{aligned} \tag{3.8}$$

zu minimieren.

Dabei ist zu beachten, daß bei der Optimierung nur die optimale Flußaufteilung berechnet wird und nicht die Wege selbst. Diese müssen auf andere Weise bestimmt und vorgegeben werden.

Eine Leitung der Übertragungskapazität $C$ wird häufig als M/M/1-Wartesystem modelliert. Als Kostenfunktion für die Kante wird dann die mittlere Anzahl der Pakete im System verwendet, die sich nach [85] zu

$$D_{ij}(F_{ij}) = \frac{F_{ij}}{C_{ij} - F_{ij}} \tag{3.9}$$

ergibt.

In [18] wird gezeigt, daß eine Lösung nur dann optimal ist, wenn der Fluß über Wege geleitet wird, bei denen der Kostenanstieg

$$\frac{\partial D(\vec{x})}{\partial x_p} = \sum_{(i,j)\in p} D'_{ij}(F_{ij}) \tag{3.10}$$

minimal ist (sog. MFDL-Pfade, minimal first derivative length).

Bei der *Frank-Wolfe-Methode* (flow deviation method) [18, 46] wird der Eingangsfluß eines Quelle-Ziel-Paars $w$ iterativ so umverteilt, daß der Fluß auf Wegen mit minimalem Kostenanstieg zunimmt und auf den restlichen Wegen zu gleichen Teilen verringert wird. Die Iterationsvorschrift lautet:

$$\vec{x}^{(\nu+1)} = \vec{x}^{(\nu)} + \alpha^{(\nu)}(\vec{\overline{x}}^{(\nu)} - \vec{x}^{(\nu)}) \tag{3.11}$$

wobei mit $\vec{\overline{x}}$ der Vektor bezeichnet wird, der entsteht, wenn der gesamte Eingangsfluß $r_w$ jedes Quelle-Ziel-Paars $w$ ausschließlich über die entsprechenden MFDL-Pfade $\overline{p}_w$ geleitet wird. Die Schrittweite $\alpha$ wird bei jedem Schritt so gewählt, daß der neue Flußvektor $\vec{x}^{(\nu+1)}$ zu den geringsten Kosten führt, die bei der Iteration in der Richtung $(\vec{\overline{x}}^{(\nu)} - \vec{x}^{(\nu)})$ möglich sind. Die Berechnung der Schrittweite ist ein eindimensionales Minimierungsproblem bezüglich $\alpha$. Approximiert man die Kostenfunktion nach Gleichung 3.9 durch eine Taylor-Reihe, so erhält man:

$$\alpha = \min\left[1, -\frac{\sum_{(i,j)} D'_{ij}(F_{ij})(\overline{F}_{ij} - F_{ij})}{\sum_{(i,j)} D''_{ij}(F_{ij})(\overline{F}_{ij} - F_{ij})^2}\right] \tag{3.12}$$

$\overline{F}_{ij}$ bezeichnet dabei den Kantenfluß, der sich entsprechend Gleichung 3.6 aus den Wegflüssen $\overline{x}_p$ ableiten läßt. Auf die Darstellung des hochgestellten Iterationsschritts $(\nu)$ wurde in obiger Gleichung verzichtet.

Beim *Gradientenprojektionsverfahren* (method of steepest descent) [124] wird die optimale Lösung ebenfalls durch eine Iteration bestimmt. Die Iteration erfolgt hier in Richtung

des Gradienten der Kostenfunktion. Die Randbedingungen werden durch Substitution bzw. Projektion berücksichtigt.

Als Iterationsvorschrift erhält man abschließend [18, 89]:

$$x_p^{(\nu+1)} = \max\left[0, x_p^{(\nu)} - \alpha\left(\sum_{(i,j)\in p} D'_{ij}(F_{ij}) - \sum_{(i,j)\in \overline{p}_w} D'_{ij}(F_{ij})\right)\right] \quad \forall\, w \in W, p \in P_w, p \neq \overline{p}_w$$

$$x_{\overline{p}_w}^{(\nu+1)} = r_w - \sum_{\substack{p\in P_w \\ p\neq \overline{p}_w}} x_p^{(\nu+1)} \qquad \forall\, w \in W \tag{3.13}$$

Die Schrittweite $\alpha$ muß geeignet gewählt werden [122]. Das Verfahren kann verbessert werden, indem der Gradient mit der inversen Hesse-Matrix gewichtet wird. Dies ist das sogenannte *Newton-Verfahren* [18, 96]. Da die Inversion der Hesse-Matrix in der Praxis häufig schwierig ist, wird in [17] mit einer Approximation gearbeitet, bei der Nichtdiagonalelemente zu Null gesetzt werden. Als Iterationsvorschrift erhält man schließlich:

$$x_p^{(\nu+1)} = \max\left[0, x_p^{(\nu)} - \alpha\left(\sum_{(i,j)\in L_p} D''_{ij}(F_{ij})\right)^{-1}\left(\sum_{(i,j)\in p} D'_{ij}(F_{ij}) - \sum_{(i,j)\in \overline{p}_w} D'_{ij}(F_{ij})\right)\right]$$
$$\forall\, w \in W, p \in P_w, p \neq \overline{p}$$

$$x_{\overline{p}_w}^{(\nu+1)} = r_w - \sum_{\substack{p\in P_w \\ p\neq \overline{p}_w}} x_p^{(\nu+1)} \qquad \forall\, w \in W \tag{3.14}$$

wobei $L_p$ die Menge der Kanten $(i,j)$ bezeichnet, die entweder zu $p$ oder zum entsprechenden MFDL-Pfad $\overline{p}_w$, aber nicht zu beiden gehören [18, 89].

Ein Problem der letzten beiden Verfahren besteht darin, eine geeignete Schrittweite für die Iteration zu finden. In [152] wurde dieses Problem genauer untersucht und eine Methode hergeleitet, mit der für jeden Iterationsschritt die beste Schrittweite berechnet werden kann. Die Berechnung erfolgt ähnlich wie bei der Frank-Wolfe-Methode. An dieser Stelle soll darauf nicht näher eingegangen werden.

Als weitere Möglichkeit, die Optimierungsaufgabe zu lösen, wird in der Literatur häufig auch die Methode der Lagrange-Relaxation verwendet [55, 56, 95]. Auf eine genauere Diskussion dieses Verfahrens soll an dieser Stelle verzichtet werden. Ein Vergleich mit den oben genannten Verfahren ist in [89] zu finden.

Für den praktischen Einsatz der oben beschriebenen Verfahren zur Verkehrslenkung ergeben sich einige Probleme. Die größte Einschränkung ist darin zu sehen, daß sie nur eine Optimierung der Flußverteilung vornehmen, d.h. die Wege müssen anderweitig berechnet

und den Verfahren als Eingangsparameter zur Verfügung gestellt werden. Die eigentliche Verkehrslenkung erfolgt dann statisch im Quellknoten, z.B. mittels multipath routing. Damit die Pakete in den weiteren Knoten den richtigen Weg nehmen, müssen spezielle Pfadkennungen vergeben werden. Diese Methode ist z.B. in [93, 132] beschrieben.

Prinzipiell ist eine verteilte Berechnung des Optimums möglich. Die dazu notwendige Bestimmung der Kostenanstiege wird in [27] diskutiert. In [49, 105, 128, 129, 130, 148] werden spezielle Protokolle für die verteilte Berechnung mit der Gradientenprojektion und in [17, 154] ein verteilter Algorithmus, der auf dem Newton-Verfahren basiert, vorgestellt. Diese haben aber den Nachteil, daß der Overhead sehr groß ist, da pro Iterationsschritt mehrere Nachrichten durch das gesamte Netz verteilt werden müssen [105, 128, 130, 154]. Außerdem sollten sowohl die Eingangsflüsse als auch die Topologie des Netzes über längere Zeit unverändert bleiben [127, 128].

Aus diesen Gründen haben die genannten Verfahren bis jetzt keinen praktischen Einsatz erfahren, zumal auch Shortest Path-Algorithmen der optimalen Lösung sehr nahe kommen können [16, 48, 109]. Optimierungsalgorithmen werden daher eher bei der Netzplanung eingesetzt (z.B. [56, 164, 165]).

### 3.3.5 Sonstige Verfahren

Die in den meisten Datennetzen eingesetzten Verfahren basieren auf den bisher vorgestellten Algorithmen. Entsprechende Verweise sind an den jeweiligen Stellen zu finden. Ein Überblick über die wichtigsten Datennetze und ihre Verkehrslenkungsalgorithmen ist z.B. in [18, 94, 125, 126] zu finden.

Besitzen die Netze eine spezielle Topologie, werden z.T. auch speziell auf diese Topologie angepasste Verfahren eingesetzt [62, 99, 100, 159].

# Kapitel 4

# Ein objektorientierter Protokollsimulator zur Untersuchung von Verkehrslenkungsalgorithmen

In diesem Kapitel soll ein Protokollsimulator vorgestellt werden, der die Untersuchung von verteilten Verkehrslenkungsalgorithmen erleichtert. Er wurde unter Anwendung objektorientierter Methoden entwickelt und basiert auf der in Kapitel 2.3 vorgestellten Simulationsbibliothek.

## 4.1 Anforderungen

Bestehende und bereits in Betrieb genommene Netze stehen für die Entwicklung und den Test neuer Verkehrslenkungsalgorithmen in der Regel nicht zur Verfügung, da die notwendige Betriebssicherheit während der Untersuchungen nicht garantiert werden kann. Die Einführung spezieller Testzeiten oder die Bereitstellung eigener Testnetze scheidet oft aus Kostengründen aus. Der Einsatz einer Computersimulation ist deshalb eine Möglichkeit, um Leistungsmerkmale neuer Algorithmen zu ermitteln. Verkehrslenkungsalgorithmen laufen häufig auf mehreren Netzknoten verteilt ab und sind dadurch schwer nachzuvollziehen. Ein geeignetes Werkzeug kann dazu dienen, die Abläufe transparent zu machen. Häufig treten auch im regulären Betrieb unerklärliche Fehlerfälle auf. Ein Simulator kann in diesen Fällen dazu eingesetzt werden, die Fehlersituationen zu reproduzieren und so helfen, die Ursache zu ergründen und eventuelle Protokollfehler zu beheben.

Durch diese vielfältigen Einsatzmöglichkeiten ergeben sich gegenüber konventionellen Protokollsimulatoren deutlich andere Anforderungen. Die wichtigsten Anforderungen, die

ein Protokollsimulator erfüllen muß, der zur Untersuchung verteilter Verkehrslenkungsalgorithmen eingesetzt wird, lauten:

- Verschiedene Netztopologien müssen ohne großen Aufwand untersucht werden können. Die Topologie eines Netzes darf nicht fest im Simulator verankert sein, sondern muß vom Anwender schnell und einfach einzugeben sein.
- Es müssen unterschiedliche Knotenarchitekturen bzw. Knotenmodelle simuliert werden können. Zusammen mit der variablen Netztopologie heißt das, daß das gesamte Simulationsmodell variabel ist und daher dynamisch aufbaubar sein muß.
- Das Protokoll muß schrittweise nachvollziehbar sein. Protokollmeldungen müssen auf Wunsch ausgegeben werden können. Um die Informationsflut einzuschränken, ist es notwendig, daß der Anwender verschiedene Filterkriterien festlegen kann.
- Die Simulation sollte interaktiv ablaufen können, d.h. der Anwender sollte jederzeit die Möglichkeit haben, in den Ablauf einzugreifen, um zusätzliche Detailinformationen abzurufen oder Parameter zu verändern.
- Verschiedene Verkehrslenkungsalgorithmen müssen leicht integrierbar sein.
- Der Simulator sollte auch nachträglich leicht erweiterbar sein, um z.B. neue Verkehrslenkungsalgorithmen aufzunehmen.
- Möglichst viele Komponenten des Simulators sollten auch für andere Zwecke wiederverwendbar sein.

## 4.2 Aufbau des Protokollsimulators

In [33, 121, 151] wurden bereits Werkzeuge zur Untersuchung von Netzen vorgestellt, die aber die oben gestellten Anforderungen bezüglich der Erweiterbarkeit und Flexibilität nicht erfüllen. Aus diesem Grund wurde ein eigener Simulator konzipiert, entworfen und in der Sprache C++ implementiert. Er ermöglicht die simulative Untersuchung der Eigenschaften von Verkehrslenkungsalgorithmen. Beim Entwurf des Simulators wurde großer Wert auf eine klare Softwarearchitektur gelegt, die es erlaubt, den gesamten Simulator oder auch nur Teile davon für andere Zwecke wiederzuverwenden.

Die Grundstruktur des Simulators ist in Abbildung 4.1 dargestellt. Im Mittelpunkt steht das zu simulierende Nachrichtennetz. Es besteht aus mehreren Knoten, die über Leitungen miteinander verbunden sind. Das Netz stellt gleichzeitig das Simulationsmodell entsprechend Abschnitt 2.3.1 dar.

Ein wesentlicher Punkt der vorliegenden Arbeit ist die Untersuchung verschiedener Netztopologien und damit verschiedener Simulationsmodelle. Aus diesem Grund ist es

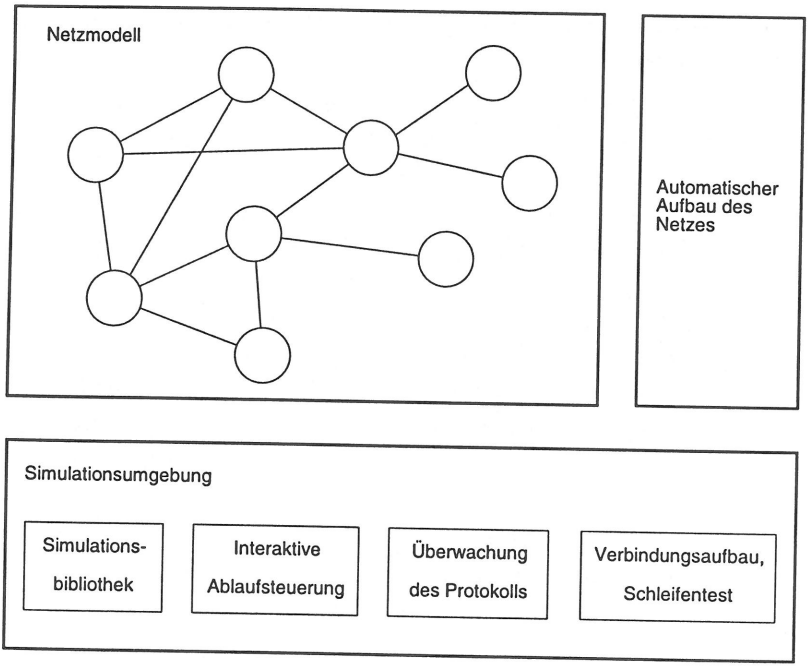



Bild 4.1: Grundstruktur des Protokollsimulators

nicht sinnvoll, ein Modell fest in den Protokollsimulator einzukodieren. Die vorgestellte Simulationsbibliothek erlaubt grundsätzlich den dynamischen Aufbau eines Modells zur Laufzeit, sie besitzt aber in ihrer bisherigen Form keine Komponente, die aus einer Netzbeschreibung automatisch ein Simulationsmodell aufbaut. Der Simulator besitzt deshalb eine zusätzliche Komponente, die eine automatische Umsetzung einer Topologiebeschreibung in ein Simulationsmodell vornimmt und dieses zur Laufzeit aufbaut.

Die dritte Komponente aus Abbildung 4.1 ist die *Simulationsumgebung*. Sie setzt sich ihrerseits aus mehreren Unterblöcken zusammen. Der erste Block repräsentiert die bereits mehrfach erwähnte objektorientierte Simulationsbibliothek. Um den besonderen Anforderungen gerecht zu werden, wurde sie um weitere Komponenten erweitert. Der normale Ablauf einer Simulation, wie er in [87] beschrieben wurde, ist: Zu Beginn werden die Simulationsparameter eingelesen, anschließend die Simulation gestartet und am Ende die Ergebnisse ausgegeben. Während der eigentlichen Simulation hat der Anwender keine Möglichkeit, in den Ablauf einzugreifen. Der Block *Interaktive Ablaufsteuerung* erweitert die Simulationsbibliothek dahingehend, daß eine Simulation nun auch interaktiv durchgeführt werden kann. Der Anwender hat die Möglichkeit, zu beliebigen Zeitpunkten

Unterbrechungspunkte (Breakpoints) zu setzen. An diesen wird die Simulation vorübergehend angehalten und die Kontrolle dem Anwender übergeben. Er kann sich dann z.B. den Systemzustand anschauen oder gezielt Änderungen vornehmen. Simulationsereignisse (vgl. Abschnitt 2.3.4) können auch im Einzelschrittbetrieb abgearbeitet werden. Diese Eigenschaft ist besonders dann von großem Nutzen, wenn der Austausch von Protokollmeldungen und deren Verarbeitung innerhalb der Knoten schrittweise nachvollzogen werden soll. Die Benutzerführung des Simulators kann auf Wunsch während der gesamten Simulation menügesteuert erfolgen.

Durch die Verwendung der Simulationsbibliothek erhält man automatisch Unterstützung bei der Erfassung einfacher Meßgrößen, wie z.B. der Anzahl der generierten und gesendeten Protokollmeldungen, Laufzeiten der Meldungen, etc. Um dem Anwender zusätzlich leistungsfähige Hilfen zur Überwachung des Protokollablaufs zu bieten, wurde ein eigenes Konzept entwickelt. Die dazu notwendigen Elemente sind im Block *Überwachung des Protokolls* enthalten.

Im letzten Unterblock der Simulationsumgebung sind Komponenten enthalten, die es erlauben, zu jedem beliebigen Zeitpunkt einen Verbindungsaufbau nachzubilden, ihn auf seinen Erfolg hin zu überwachen, Schleifentests durchzuführen und die verteilte Verkehrslenkungsinformation in den verschiedenen Knoten auf Konsistenz zu überprüfen.

Auf den Aufbau und die zugrundeliegenden Entwurfskonzepte der wichtigsten Komponenten des Simulators wird in den nachfolgenden Abschnitten ausführlicher eingegangen.

## 4.3 Modellierung

Wie bereits im vorigen Abschnitt erwähnt wurde, bildet das Netz, bestehend aus Knoten und Leitungen, das eigentliche Simulationsmodell. Die Knoten und Leitungen sind dabei hierarchisch aufgebaute Modellkomponenten, die noch detaillierter modelliert werden müssen. In Abbildung 4.2 ist ein detailliertes Modell eines Knotens und eines Teils seiner Leitungen dargestellt.

Eine Leitung wird für jede Übertragungsrichtung durch ein Verzögerungsglied modelliert, das die Laufzeit eines Pakets auf der Leitung repräsentiert. Nachdem die Pakete an einem Knoten angekommen sind, durchlaufen sie die Schichten 1 und 2. Um schnelle Reaktionszeiten auf Änderungen innerhalb des Netzes zu erreichen, werden bei vielen realen Implementierungen Verkehrslenkungsprotokollpakete innerhalb des Knotens priorisiert behandelt [65, 97, 158]. Eine Unterscheidung zwischen diesen Protokollpaketen und den restlichen Paketen, die im folgenden auch als „Nutzpakete“ bezeichnet werden, kann frühestens oberhalb der Schicht 2 erfolgen. Die priorisierte Behandlung wird in dem Modell durch einen Prioritätsprozessor durchgeführt. Verkehrslenkungsprotokollpakete, oder

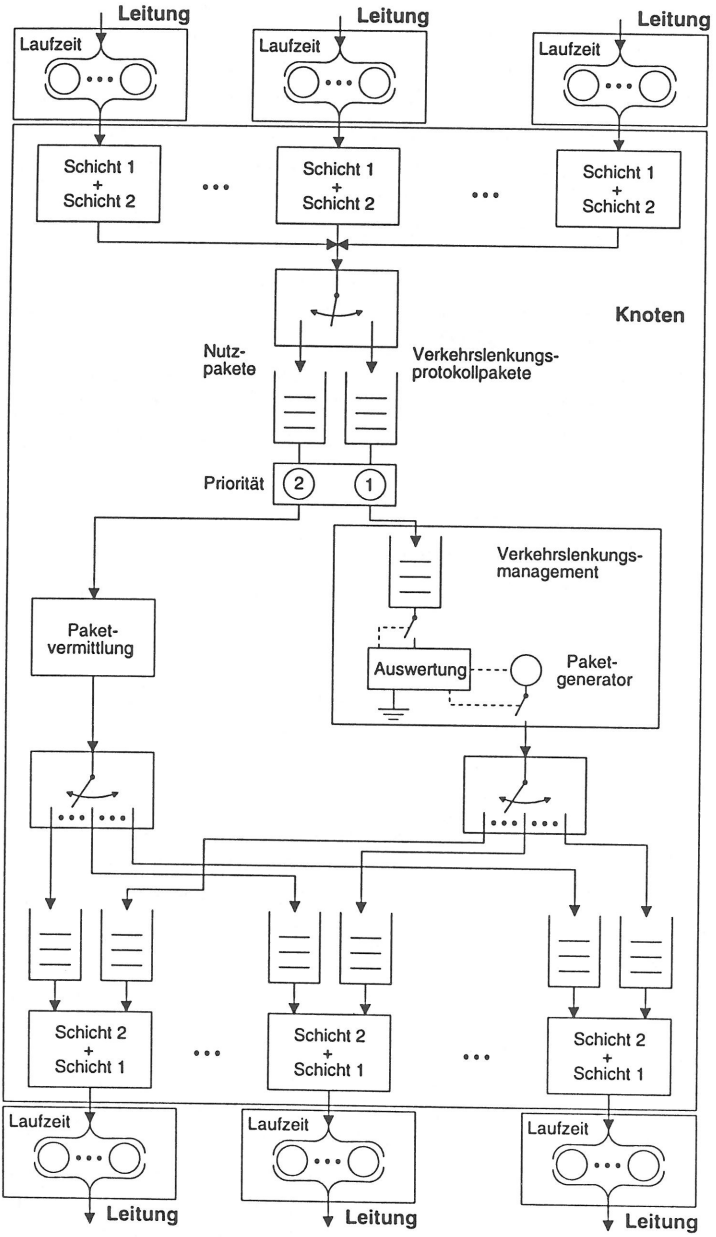



Bild 4.2: Modellierung eines Knotens und der Leitungen

kurz Protokollpakete, besitzen die Priorität 1 und werden immer bevorzugt behandelt. Nutzpakete besitzen die niedrigere Priorität 2. Nach dem Prozessor teilt sich der Pfad der Nutz- und der Protokollpakete. Nutzpakete gelangen zu einem Block *Paketvermittlung*, in dem die Funktionen für die Vermittlung von Paketen, den Auf- und Abbau von Verbindungen, etc. enthalten sind. Auf eine genauere Modellierung des Blocks wurde an dieser Stelle verzichtet. Nachdem der Block durchlaufen wurde, werden die Pakete in den Ausgangspuffer einer Leitung eingetragen. Vor der Übertragung auf der Leitung durchlaufen sie nochmals die Schichten 2 und 1.

In realen Implementierungen für das Verkehrslenkungsmanagement wird nicht für jedes ankommende Verkehrslenkungsprotokollpaket sofort ein neues Paket mit den entsprechenden Daten generiert und versendet. Um die Zahl der Protokollpakete zu reduzieren, werden diese gesammelt, redundante Pakete entfernt und dann erst ein neues Protokollpaket erzeugt und verschickt [163]. Häufig erfolgt der Austausch von Protokollinformation auch in regelmäßigen Abständen [102, 103]. Dieses Aufsammeln der Pakete wird durch eine Warteschlange modelliert, in die die Protokollpakete nach der Bearbeitung durch den Prioritätsprozessor eingetragen werden. Der Block *Auswertung* wertet die wartenden Pakete aus, löscht alte oder doppelte Pakete und steuert die Erzeugung neuer Protokollpakete. Er steuert auch das Versenden dieser Pakete, indem er zu bestimmten Zeitpunkten den eingezeichneten Schalter schließt. Schließlich werden die Protokollpakete in die entsprechenden Ausgangspuffer eingetragen. Auch an dieser Stelle erfolgt in der Regel eine bevorzugte Behandlung durch die Schichten 2 und 1. Aus diesem Grund enthält das Modell zwei Warteschlangen: eine für Nutzpakete und eine für Protokollpakete.

Für die Funktion des Verkehrslenkungsprotokolls sind nur die Protokollpakete von Bedeutung. Betrachtet man den Weg eines ankommenden Protokollpakets von der Ankunft am Knoten bis zum Block *Auswertung*, stellt man folgendes fest: Auf Grund der Priorisierung haben Nutzpakete nur einen sehr geringen Einfluß auf die Warte- und Bearbeitungszeiten der Protokollpakete vor dem bzw. im Prioritätsprozessor. Anschließend werden alle ankommenden Pakete in der Warteschlange vor der Auswertung gesammelt und gemeinsam bearbeitet. Die Durchlaufzeiten durch die Schichten 1, 2 und den Prioritätsprozessor sind gegenüber dieser Wartezeit vernachlässigbar. Auf der Sendeseite werden die Protokollpakete in den Schichten 2 und 1 ebenfalls priorisiert. Die Durchlaufzeit durch diese Schichten wird deshalb durch den Nutzpaketverkehr nicht wesentlich beeinflußt. Dominierend für das zeitliche Verhalten ist der Takt, in dem die *Auswertung* empfangene Pakete bearbeitet, neue erzeugt und versendet. Der Nutzpaketpfad kann deshalb vollständig vernachlässigt werden. Durch diese Vereinfachungen des Modells wird der asynchrone Austausch synchronisiert. Diese vereinfachende Annahme eines synchronen Nachrichtenaustauschs wird in der Literatur häufiger getroffen, um verschiedene Algorithmen miteinander vergleichen zu können [9, 15, 163].

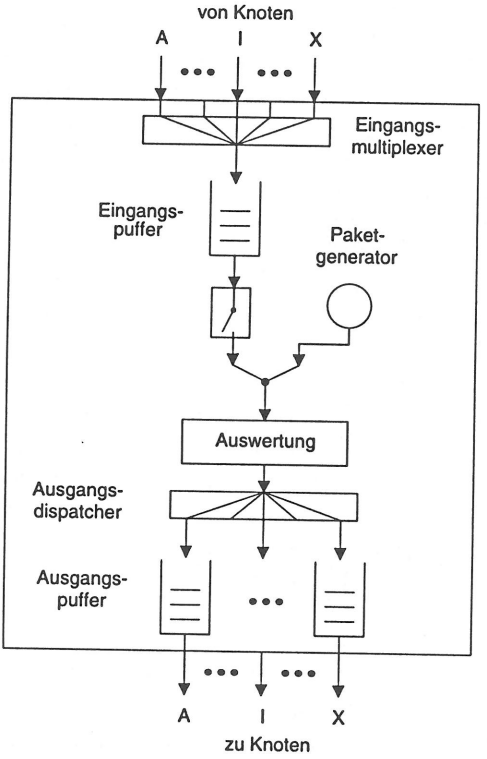



Bild 4.3: Vereinfachtes Knotenmodell

Das vereinfachte Knotenmodell, bei dem nur der Protokollpaketpfad betrachtet wird, ist in Abbildung 4.3 dargestellt. Alle ankommenden Pakete werden über einen Multiplexer einem Eingangspuffer zugeführt. Dieser dient zur Serialisierung der Bearbeitung. In der Komponente *Auswertung* ist das eigentliche Verkehrslenkungsprotokoll realisiert. Hier werden die Pakete ausgewertet, Verkehrslenkungstabellen berechnet und gegebenenfalls neue Protokollpakete erzeugt und versendet. Sie entspricht im wesentlichen der Auswertungskomponente aus Abbildung 4.2. An jedem Ausgang befindet sich ein Ausgangspuffer, in den die Pakete eingereiht werden, bevor sie über eine Leitung übertragen werden. Der Paketgenerator dient dazu, interne Steuerpakete zu erzeugen, die eine Zustandsänderung einer Leitung anzeigen, d.h. der Generator erzeugt immer dann ein Paket, wenn eine Leitung ausfällt, sie wieder in Betrieb genommen wird oder sich ihre Länge ändert.

Das vereinfachte Modell einer Leitung ist in Abbildung 4.4 dargestellt. Die beiden Übertragungsrichtungen werden getrennt voneinander betrachtet. Da die Pakete in den

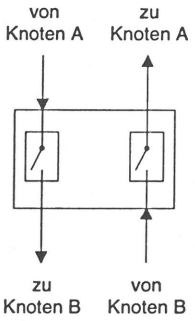



Bild 4.4: Vereinfachtes Leitungsmodell

Knoten gesammelt und gemeinsam bearbeitet werden und die Übertragungszeit über die Leitung wesentlich kürzer als der Abstand zwischen zwei Bearbeitungszeitpunkten des Knotens ist, kann jede Richtung durch einen Schalter modelliert werden. Sind alle in den Eingangspuffern der Knoten wartenden Pakete bearbeitet worden, werden die Schalter in den Leitungen geschlossen und die zwischenzeitlich neu erzeugten Pakete, die in den Ausgangspuffern warten, übertragen. Die Modellierung der Leitungen durch Schalter ist notwendig, um zu verhindern, daß Pakete, die in einem Knoten erzeugt und versendet werden, noch während des gleichen Taktzykluses beim Nachbarknoten bearbeitet werden. Dies hätte dann zur Folge, daß ein Paket unter unglücklichen Bedingungen innerhalb einer Taktperiode durch das gesamte Netz verteilt werden könnte.

## 4.4 Flexible Eingabe der Netztopologie

Sowohl bei analytischen als auch bei simulativen Untersuchungen von Protokollen ist die Netztopologie ein wichtiger Parameter. Der Aufwand für die Untersuchung kann deutlich reduziert werden, wenn unterschiedliche Topologien einfach und flexibel in ein Analyse- oder Simulationsprogramm eingegeben werden können.

Für die benutzerfreundliche Eingabe von Netzen sind grundsätzlich zwei Ansätze denkbar: zum einen die Beschreibung durch eine Eingabedatei oder zum anderen die Eingabe durch eine graphische Oberfläche. Die Vorteile letzterer Lösung liegen in einer ansprechenden Darstellung. Klare Nachteile sind aber die erschwerte Portierung auf mehrere Plattformen und unter Umständen ein höherer Aufwand bei der Eingabe sehr großer Netze. Insbesondere dort können sehr viele „Maus"-Aktionen bei der Eingabe notwendig werden. Die Vorteile einer graphischen Oberfläche werden dadurch relativiert. Aus diesem Grund wurde eine textuelle Beschreibungsform des Netzes bevorzugt. Die Eingabe kann

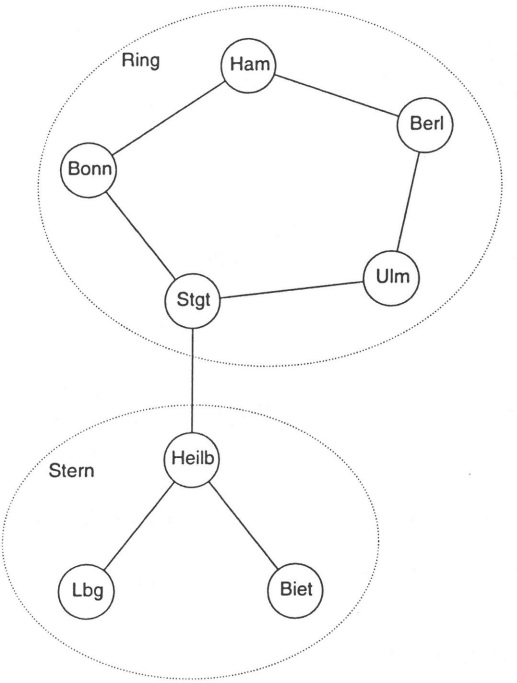
sehr leicht um eine graphische Oberfläche erweitert werden. Die textuelle Beschreibung dient dann als Bindeglied zwischen der Oberfläche und dem eigentlichen Kern der Analyse oder Simulation. Dies hat den zusätzlichen Vorteil, daß Abhängigkeiten zwischen beiden reduziert werden.

Bei der Eingabe eines Netzes wurden folgende allgemeine Gedanken umgesetzt, die es erlauben, auch sehr große Netze schnell und flexibel einzugeben: Ein Netz setzt sich aus verschiedenen Subnetzen zusammen, die wiederum aus Knoten und Verbindungsleitungen zwischen diesen bestehen. Ein Subnetz kann, aber muß nicht, eine regelmäßige Struktur aufweisen. Zur Kennzeichnung wird jedem Knoten ein Name und eine eindeutige Identifikationsnummer (ID) zugeordnet. Beliebige Knoten können durch eine Leitung verbunden werden. Die Knoten dürfen dabei auch zu verschiedenen Subnetzen gehören. Sowohl Knoten als auch Leitungen können Parameter besitzen, die ihren Aufbau und ihre Eigenschaften näher beschreiben.

Analog dem beschriebenen hierarchischen Aufbau eines Netzes aus Subnetzen weist die Datei zur Eingabe der Topologie eine geschachtelte Blockstruktur auf. Jeder Block wird durch ein Schlüsselwort eingeleitet. Er kann weitere Blöcke oder Schlüsselworte enthalten, denen Werte zugewiesen werden. Innerhalb des Blocks der obersten Ebene können Schlüsselworte für eine der Grundtopologien aus Kapitel 2.1.1, Schlüsselworte zum Verbinden von Knoten sowie zum Löschen von bereits bestehenden Verbindungen stehen. Jedes dieser Schlüsselworte leitet einen untergeordneten Block ein, der die Eigenschaften des Subnetzes oder der Verbindung genauer beschreibt. Innerhalb des untergeordneten Blocks sind Schlüsselworte für die Anzahl der Knoten, deren Namen sowie den Start- bzw. Endknoten einer Verbindung bekannt. Zusätzlich zu den vordefinierten Schlüsselworten kann der Anwender eigene zur Laufzeit registrieren lassen, die zur Unterscheidung verschiedener Knoten- bzw. Leitungstypen dienen. Sie leiten ebenfalls einen Block ein, in dem die typ- bzw. modellspezifischen Parameter stehen.

Abbildung 4.5 zeigt eine Beispieltopologie und die zugehörige Eingabe. Sowohl die Knoten als auch die Leitungen besitzen dabei einen zusätzlichen Parameter, der eine Verzögerungszeit innerhalb des Knotens bzw. die Leitungslänge angibt.

In dem Klassendiagramm in Abbildung 4.6 sind die wichtigsten Klassen dargestellt, die sich mit der Eingabe befassen. Für das Einlesen der oben beschriebenen Topologiedaten sind vier Klassen verantwortlich. Die Auftrennung in verschiedene Klassen spiegelt ein Grundprinzip des objektorientierten Entwurfs wieder. Jede Klasse hat eine begrenzte, genau definierte Aufgabe zu erfüllen. Kompliziertere Funktionen werden durch Zusammenarbeit mehrerer Klassen erbracht und nicht durch Erweitern der Aufgaben einer Klasse. Alle diese Klassen sind von einer gemeinsamen Basisklasse TParserObject abgeleitet, von der sie die Fähigkeit erben, Schlüsselworte zu erkennen und ihnen Werte zuzuweisen.




```
Ring {
   Number=5;
   Name="Stgt";
   Name="Bonn";
   Name="Ham";
   Name="Berl";
   Name="Ulm";
   Node {
      Delay=5;
   }
   Link {
      Length=7;
  }
}

Star {
   Number=3;
   Name="Heilb";
   Name="Lgb";
   Name="Biet";
   Node {
      Delay=15;
   }
   Link {
      Length=16;
   }
}

Connect {
   Start="Heilb";
   End="Stgt";
   Link {
      Length=4;
   }
}
```

Bild 4.5: Beispielnetz und zugehörige Eingabedatei

Die Klasse TNETWORKPARSER ist für das Einlesen eines Netzes verantwortlich, das aus mehreren Subnetzen bestehen kann. Sie kennt Schlüsselworte für die elementaren Grundtopologien Ring, Bus, etc. Wird ein solches erkannt, wird die weitere Bearbeitung an ein Objekt der Klasse TSUBNETPARSER delegiert. Dort sind Schlüsselworte für die Anzahl der Knoten eines Subnetzes, deren Namen usw. definiert. Zusätzlich werden weitere Schlüsselworte dynamisch verwaltet, die einen speziellen Knoten- oder Leitungstyp charakterisieren.

Die Parameter eines Knotens oder einer Leitung werden von jeweils einer eigenen Klasse, die von TNODEPARSER bzw. TLINKPARSER abgeleitet sein muß, eingelesen und behandelt. Wird eine Instanz eines neuen Knoten- oder Leitungsparsers erzeugt, registriert sich diese automatisch mit ihrem Schlüsselwort beim Subnetzparser. Wird dieses während

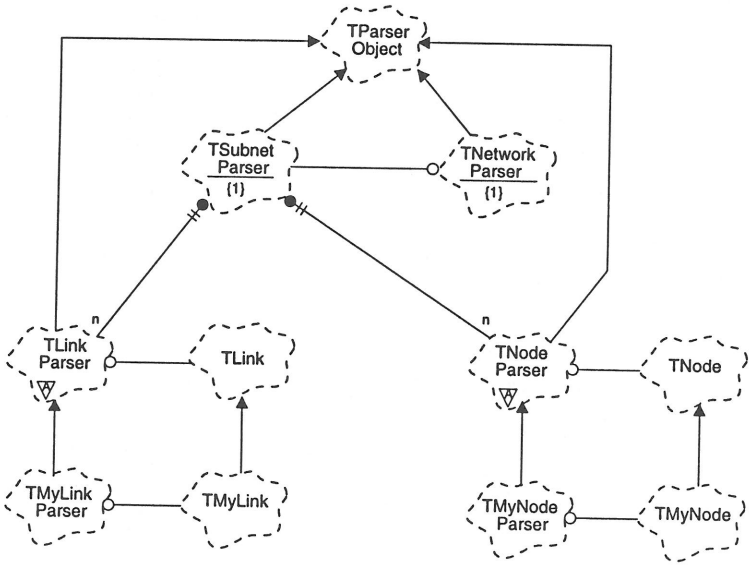



Bild 4.6: Klassendiagramm der wichtigsten Klassen für das Einlesen eines Netzes

des Einlesens erkannt, wird die weitere Bearbeitung an den entsprechende Knoten- bzw. Leitungsparser delegiert.

Bei den Klassen TLINKPARSER und TNODEPARSER handelt es sich um abstrakte Basisklassen, die eine gemeinsame Verwaltung aller abgeleiteten Klassen ermöglichen. Sie stellen außerdem über die Methoden CreateNode bzw. CreateLink eine allgemeine Schnittstelle bereit, über die letztendlich konkrete Instanzen von Knoten bzw. Leitungen mit den entsprechenden Parametern erzeugt werden können. Sie dienen als eine Art virtueller Konstruktor oder Objektfabrik (siehe auch Abschnitt 4.7.1). Wie man aus Abbildung 4.6 erkennt, existiert parallel zur Hierarchie der Knoten- und Leitungsparserklassen eine entsprechende Hierarchie von Knoten- bzw. Leitungsklassen. Diese parallele Klassenhierarchie ist typisch für das Entwurfsmuster Objektfabrik.

Durch die Möglichkeit, neue Schlüsselworte dynamisch registrieren zu lassen, und die Verwendung einer Objektfabrik wird eine weitgehende Entkopplung der Eingabe von dem restlichen Programm erreicht. Dadurch ist es möglich, die Eingabekomponente ohne Änderungen für alle Programme wiederzuverwenden, bei denen eine Topologie eingelesen werden muß. Dies können sowohl Analyse- als auch Simulationsprogramme sein.

Die Integration der Topologieeingabe in ein Programm ist mit minimalem Aufwand möglich. Der Anwender muß dazu nur die Klassen, die in seinem Programm die Knoten

bzw. Leitungen repräsentieren, von TNODE bzw. TLINK ableiten. Zusätzlich definiert er jeweils eine Klasse für einen Knoten- bzw. Leitungsparser, bei denen die CreateNode bzw. CreateLink Methode überschrieben und ein entprechender Knoten bzw. eine Leitung erzeugt wird. Die Integration erfolgt abschließend durch dynamisches Registrieren der Knoten- und Leitungsparser beim Subnetzparser. Modifikationen von bestehendem Code sind nicht notwendig.

## 4.5 Objektorientierte Dekomposition

### 4.5.1 Übertragung von Steuerinformation

Bevor auf die Dekomposition der verschiedenen Verkehrslenkungsalgorithmen und ihre Einbettung in das Knotenmodell eingegangen wird, soll kurz auf ein Problem hingewiesen werden, das bisher vernachlässigt wurde. Dieses Problem besteht in der Übertragung von Steuerinformation zwischen der Modellkomponente, in der die Vermittlungsfunktionen realisiert sind, und anderen Modellkomponenten. Beim Flooding z.B. darf ein Paket nicht mehr zu dem Nachbarknoten gesendet werden, von dem es empfangen wurde. Dazu muß die Auswertekomponente, in der das Verkehrslenkungsprotokoll realisiert ist, wissen, wer ein Paket gesendet hat. Diese Information ist am Eingangsmultiplexer des Knotens bekannt, da dieser die Nachbarn zu den entsprechenden Eingängen zuordnen kann. Umgekehrt wird in der Auswertekomponente bestimmt, zu welchem Nachbar ein Paket gesendet wird. Benötigt wird diese Information im Ausgangsdispatcher.

In der verwendeten Simulationsbibiliothek sind keine unmittelbaren Hilfsmittel zur transparenten Übermittlung von Steuerinformation über mehrere dazwischenliegende Modellkomponenten hinweg vorhanden. An dieser Stelle läßt sich ein entscheidender Vorteil des objektorientierten Ansatzes ausnutzen. Das Handshake-Protokoll, mit dem die Pakete zwischen den einzelnen Komponenten übertragen werden, ist unabhängig vom tatsächlichen Typ eines Pakets. Die zughörige Klasse muß nur von TMESSAGE abgeleitet sein.

Die Tatsache, daß eine Komponente nicht für einen bestimmten Pakettyp maßgeschneidert ist, läßt sich sehr elegant zur Übertragung von Steuerinformation einsetzen. Man kann sich eines Prinzips bedienen, wie es auch im OSI-Referenzmodell zum Austausch von Daten zwischen Protokollinstanzen verwendet wird. Dort hängt jede Schicht ihre Steuerinformation (PCI, protocol control information) an die Nutzinformation an und übergibt beides als neue Nutzinformation an die tieferliegende Schicht. Im Sender wird ein Paket beim Durchlaufen der Schichten also schrittweise „eingepackt“ und im Empfänger entsprechend „ausgepackt“.

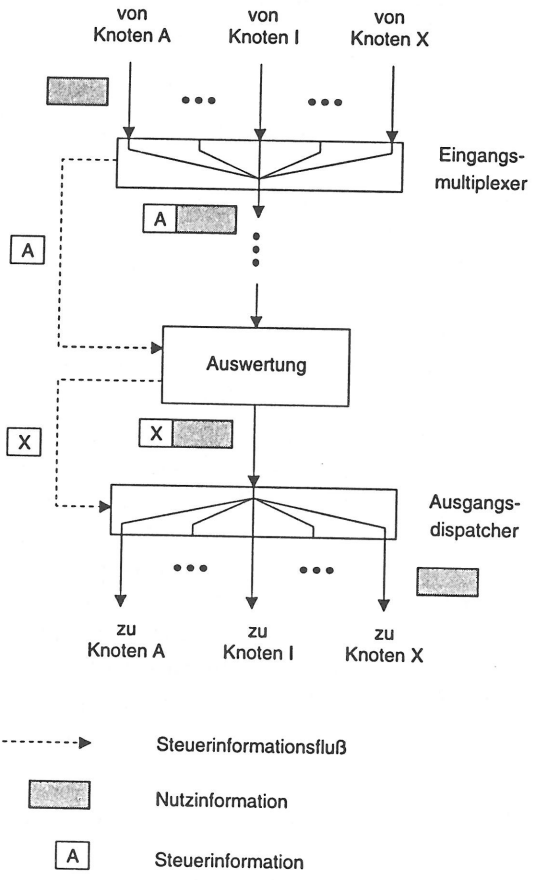



Bild 4.7: Übertragung von Steuerinformation zwischen Modellkomponenten

Abbildung 4.7 verdeutlicht dieses Prinzip für die Übertragung von Steuerinformation zwischen Modellkomponenten. Im Eingangsmultiplexer wird ein neues Paket erzeugt, das aus dem empfangenen Paket und der zugehörigen Steuerinformation besteht. Unter Ausnutzung der Polymorphie kann dieses neue Paket problemlos über alle Modellkomponenten übertragen werden. In der Auswertekomponente wird das ursprüngliche Paket ausgepackt und die Steuerinformation ausgewertet. Will die Auswertekomponente ein Paket versenden, fügt sie diesem auf die gleiche Weise die Information über den Nachbar hinzu, an den das Paket gesendet werden soll. Diese Steuerinformation wird dann vom Ausgangsdispatcher ausgewertet.

### 4.5.2 Algorithmen

Die zentrale Aufgabe des Simulators ist die Untersuchung und der Vergleich verschiedener Verkehrslenkungsalgorithmen bzw. -protokolle. Wie in Abschnitt 4.3 bereits kurz erwähnt wurde, ist die Verarbeitung der Protokollmeldungen in einer eigenen Modellkomponente gekapselt. Eine Möglichkeit, die verschiedenen Protokolle in den Simulator zu integrieren, besteht darin, voneinander unabhängige Modellkomponenten zu implementieren. Eine Zielsetzung des Simulators ist aber, den Aufwand für die Untersuchung neuer Algorithmen so gering wie möglich zu halten. Die Modellkomponenten für die verschiedenen Protokolle weisen genug Gemeinsamkeiten auf, die es lohnt, zu analysieren und in eine Klassenhierarchie umzusetzen.

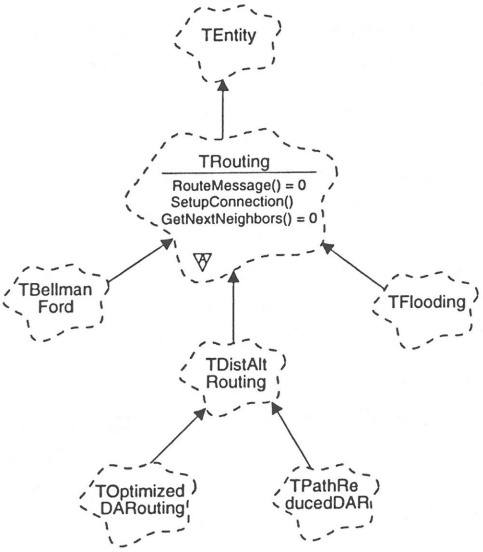



Bild 4.8: Klassendiagramm der Verkehrslenkungsalgorithmen

Das Klassendiagramm in Abbildung 4.8 zeigt die wichtigsten Zusammenhänge. Die Klasse TROUTING repräsentiert eine Modellkomponente, die die in diesem Zusammenhang relevanten Vermittlungsfunktionen der Schicht 3 enthält. Dies sind die Vermittlung eines Pakets, die Verarbeitung von Protokollinformation und der Aufbau von Verbindungen im Falle der verbindungsorientierten Kommunikation. Für diese Aufgaben besitzt die Klasse entsprechende Methoden, von denen die wichtigsten in dem Klassendiagramm aufgeführt sind. Da TROUTING auch eine Modellkomponente ist, wird sie von der Basisklasse aller Modellkomponenten TENTITY abgeleitet.

Die Klasse selbst realisiert kein bestimmtes Verkehrslenkungsprotokoll. Sie faßt die Gemeinsamkeiten aller Verkehrslenkungsverfahren zusammen und stellt eine allgemeine Schnittstelle zur Verfügung. Konkrete Protokolle werden durch Ableiten von TROUTING und entsprechendes Überschreiben der Methoden RouteMessage und GetNextNeighbors realisiert. In der ersten Methode werden ankommende Pakete ausgewertet und, falls es sich um Protokollpakete handelt, entsprechend bearbeitet. Die zweite Methode liefert einen oder mehrere Nachbarn, an die ein Paket weitergeleitet werden soll. Diese Knoten können entweder durch Aufruf eines Algorithmuses oder durch Lesen einer Verkehrslenkungstabelle bestimmt werden.

An dieser Stelle wird die Polymorphie sehr stark ausgenutzt. Innerhalb des Simulators werden nur Referenzen oder Zeiger der gemeinsamen Basisklasse aller Verkehrslenkungsverfahren verwendet. Diese werden mit Objekten von abgeleiteten Klassen initialisiert (vgl. auch 4.5.3). Auf diese Weise lassen sich die Verfahren sehr leicht gegeneinander austauschen, ohne daß bestehender Code modifiziert werden muß.

Zusätzlich können in der Klasse TROUTING bereits Funktionen realisiert werden, die unabhängig vom Verkehrslenkungsprotokoll sind. Dies sind z.B. die Realisierung des Handshake-Protokolls, das für den Empfang und das Aussenden von Paketen notwendig ist, die Auswertung der Steuerinformation des Eingangsmultiplexers oder das Hinzufügen von Steuerinformation für den Ausgangsdispatcher. Durch das Zusammenfassen dieser Gemeinsamkeiten in einer Basisklasse läßt sich der Aufwand, der notwendig ist, um neue Verkehrslenkungsverfahren in den Simulator zu integrieren, erheblich reduzieren. Der entstehende Code ist kürzer und damit weniger fehleranfällig.

In dem Klassendiagramm sind beispielhaft einige abgeleitete Klassen eingezeichnet. Sie entsprechen den Algorithmen, die später in Kapitel 6 verglichen werden. Objekte, die das in Kapitel 5.4 beschriebene Protokoll in seiner Grundversion realisieren, sind Instanzen der Klasse TDISTALTROUTING. Die verschiedenen Varianten des Protokolls, die in Kapitel 5 entwickelt und vorgestellt werden, weichen zum Teil nur an wenigen Stellen von dieser Grundversion ab. Isoliert man diese Stellen in einer eigenen Methode, läßt sich durch Ableiten einer neuen Klasse und Überschreiben der jeweiligen Methoden der Aufwand für die Untersuchung mehrerer Varianten nochmals deutlich reduzieren.

### 4.5.3 Knoten- und Leitungsmodelle

Ein Entwicklungsziel des Simulators war es, dem Anwender in jeder Beziehung eine hohe Flexibilität zu bieten. Wie unterschiedliche Netztopologien bzw. Verkehrslenkungsalgorithmen berücksichtigt werden können, wurde in den Abschnitten 4.4 und 4.5.2 vorgestellt. Ein letzter Freiheitsgrad besteht nun noch in der Berücksichtigung unterschiedlicher Knoten- und Leitungsmodellierungen sowie der Integration der Algorithmen in ein

bestehendes Modell. Im Abschnitt 4.3 wurden verschieden detaillierte Modelle für Knoten und Leitungen vorgestellt. Erkennt man im Laufe einer Untersuchung, daß einzelne Teile anders oder detaillierter modelliert werden müssen, sollte dies einfach und weitestgehend ohne große Änderungen des bestehenden Codes möglich sein.

Der Simulator definiert deshalb auch im Bereich der Knoten- und Leitungsmodellierung mehrere Abstraktionen mit eindeutigen Schnittstellen, die als Basis für Erweiterungen dienen können. Abbildung 4.9 zeigt einen Ausschnitt aus dem Klassendiagramm, das die Knotenmodellierung betrifft.

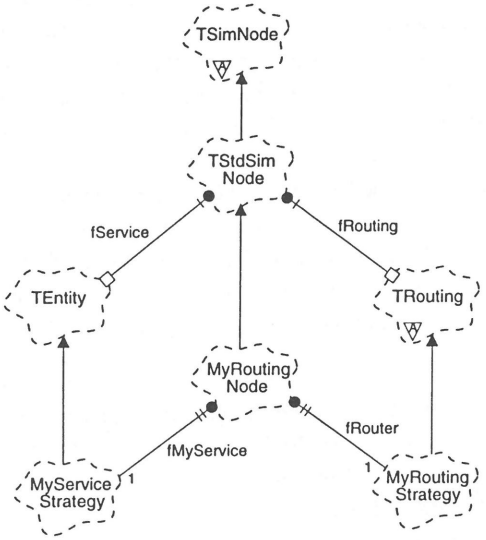



Bild 4.9: Klassenhierarchie für Knotenmodelle

Basisklasse für alle Knoten ist die Klasse TSIMNODE. Sie vereinigt die Eigenschaften einer Modellkomponente mit denen eines einlesbaren Knotens (vgl. Abschnitt 4.4). Da an dieser Stelle noch kein konkretes Knotenmodell berücksichtigt wird, ist sie eine abstrakte Basisklasse. Das Modell aus Abbildung 4.3 ist in der Klasse TSTDSIMNODE realisiert. Ein Objekt dieser Klasse enthält die meisten Modellkomponenten direkt als entsprechende Felder, so daß sich auch hier wieder eine zum hierachischen Aufbau des Knotens korrespondierende Objekthierarchie ergibt (vgl. Abschnitt 2.3.2). Um das Diagramm übersichtlicher zu halten, wurden nicht alle Beziehungen zwischen der Klasse TSTDSIMNODE und den Klassen der enthaltenen Modellkomponenten dargestellt, sondern nur die Beziehungen zu der Komponente mit der Verkehrslenkung bzw. der Bedienstrategie innerhalb des Knotens, auf die nachfolgend genauer eingegangen wird.

In dem vereinfachten Modell des Knotens sind im wesentlichen zwei Komponenten enthalten, die häufiger geändert oder ausgetauscht werden müssen. Dies ist zum einen die Verkehrslenkung und zum anderen die Bedienstrategie innerhalb eines Knotens. Sollen verschiedene Verkehrslenkungsverfahren miteinander verglichen werden, benötigt man Knotenobjekte, die unterschiedliche Verkehrslenkungskomponenten enthalten. In dem vorgestellten Modell arbeiten die Knoten synchron. Soll das Modell schnell auf asynchrone Arbeitsweise umgestellt werden, muß die synchrone Bedienung mittels des Schalters z.B. durch ein Verzögerungsglied oder einen Prozessor ersetzt werden.

Um eine hohe Flexibilität und leichte Erweiterbarkeit zu erhalten, sind sowohl die Komponente für die Verkehrslenkung als auch für die Bearbeitung innerhalb eines Knotens nicht direkt in einem Objekt der Klasse TStdSimNode enthalten. Im Falle der Verkehrslenkung enthält diese vielmehr einen Zeiger auf die abstrakte Basisklasse aller Verkehrslenkungsstrategien TRouting. Dieser Zeiger kann später mit einem beliebigen Objekt initialisiert werden, das die konkrete Strategie verkörpert. Entsprechend enthält TStdSimNode einen Zeiger auf die Basisklasse aller Modellkomponenten, der später mit einer beliebigen Bedienkomponente initialisiert werden kann. Die Bereitstellung der Komponenten für die Verkehrslenkungs- und Bedienstrategie erfolgt durch Ableiten neuer Klassen von TStdSimNode. Diese enthalten entsprechende Objekte direkt als Felder und sorgen für die richtige Initialiserung der Zeiger in der Basisklasse.

Der entscheidende Vorteil, den man durch diese mehrstufige Klassenhierarchie erhält, liegt in einer sehr hohen Flexibilität bei der Berücksichtigung neuer Modelle und Verkehrslenkungstrategien. Dem Anwender werden je nach Umfang der beabsichtigten Erweiterung mehrere Ausgangspunkte zur Verfügung gestellt. Soll der Simulator mit einem vollständig neuen Knotenmodell arbeiten, kann eine neue Klasse von TSimNode abgeleitet werden. Die Schnittstelle zu den Knoten bleibt dabei erhalten. Sollen nur neue Verkehrslenkungsstrategien oder Bearbeitungsstrategien in Verbindung mit dem Modell aus Abbildung 4.3 verwendet werden, ist dies mit minimalem Aufwand möglich. Der Anwender muß lediglich eine Klasse von TStdSimNode ableiten und dort jeweils ein Strategieobjekt bereitstellen.

Für die Leitungsmodelle existiert eine ähnliche Klassenhierarchie. Eine abstrakte Basisklasse stellt eine allgemeine Schnittstelle zur Verfügung, die es ermöglicht, Leitungen mit ihren Parametern einzulesen und Knoten über Leitungen zu verbinden. Abgeleitete Klassen berücksichtigen dann konkrete Modelle. Da die Klassenhierarchie der Hierarchie bei den Knoten sehr ähnlich ist, soll an dieser Stelle auf eine detaillierte Diskussion verzichtet werden.

Abschließend soll in diesem Abschnitt der Ablauf der Simulation und der Weg eines Pakets durch die Knoten- und Leitungsmodelle kurz erläutert werden. In Abbildung 4.10

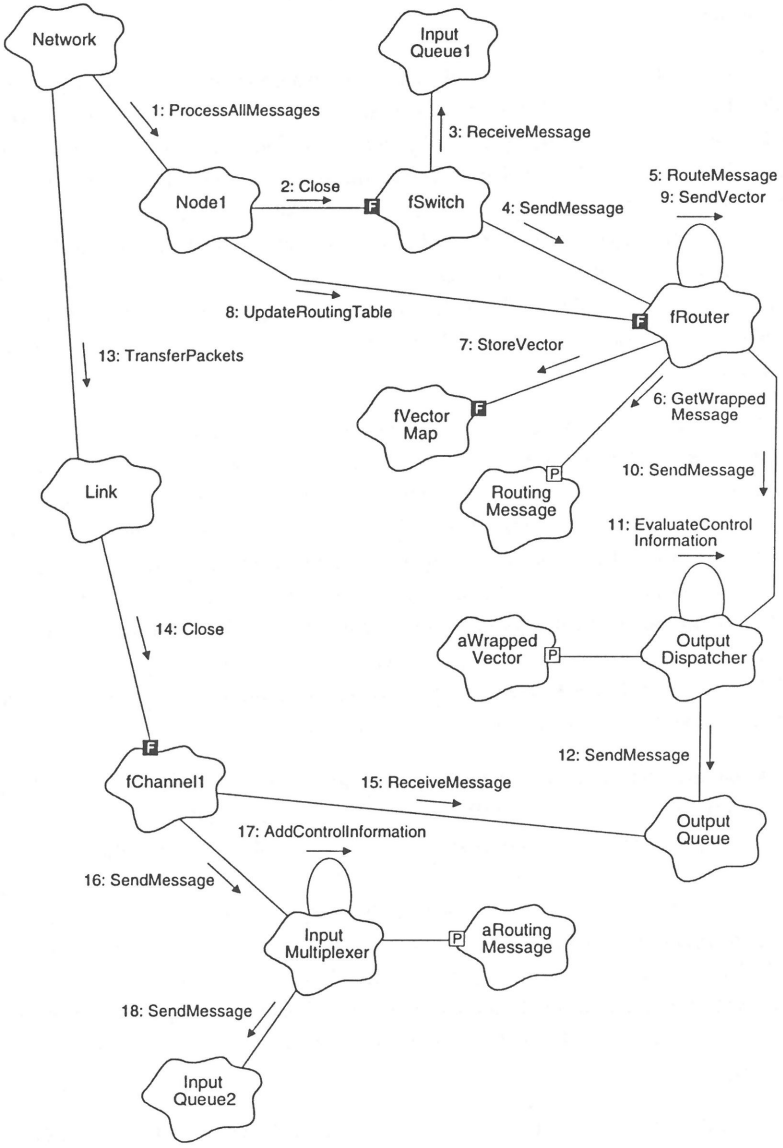



Bild 4.10: Abläufe während eines Taktzykluses

sind die wichtigsten Abläufe während eines Taktzykluses in Form eines Objektdiagramms dargestellt.

Auf der obersten Hierarchiestufe der Modellkomponenten ist das Netz. Zu Beginn jedes Takts sendet das Objekt die Meldung ProcessAllMessages der Reihe nach an alle Netzknoten. In der Abbildung ist dies beispielhaft für den Knoten 1 dargestellt. Bei Erhalt dieser Meldung schließt der Knoten seinen internen Schalter solange, bis alle wartenden Meldungen von der Eingangswarteschlange zur Verkehrslenkungskomponente übertragen sind. Die Übertragung von Paketen zwischen einzelnen Modellkomponenten ist in der Abbildung vereinfacht dargestellt. Das Handshake-Protokoll über Portobjekte wurde auf der Empfangsseite zu der Methode ReceiveMessage und auf der Sendeseite zu der Methode SendMessage zusammengefaßt. Auf die Darstellung der entsprechenden Portobjekte wurde verzichtet.

Nachdem die Verkehrslenkungskomponente ein Paket empfangen hat, wird die interne Methode RouteMessage aufgerufen. In dieser Methode ist der eigentliche Verkehrslenkungsalgorithmus bzw. das Protokoll zum Austausch der Verkehrslenkungsinformationen realisiert (vgl. Abschnitt 4.5.2). Innerhalb dieser Methode wird das empfangene Paket als erstes ausgepackt. Die Verkehrslenkung erhält so den Sender des Pakets und die eigentliche Protokollinformation (siehe Abschnitt 4.5.1). In der Abbildung sei angenommen, daß es sich bei dem Verkehrslenkungsverfahren um ein Distance-Vector-Protokoll handelt, so daß man nach dem Auspacken mittels der Methode GetWrappedMessage einen Distanzvektor erhält. Diesen Distanzvektor speichert die Verkehrslenkungskomponente intern zusammen mit einem Verweis auf den sendenden Nachbarknoten ab.

Wenn alle Pakete in der Eingangswarteschlange eines Knotens bearbeitet sind, sendet der Knoten die Meldung UpdateRoutingTable an die Verkehrslenkung. Dadurch wird die Neuberechnung einer Verkehrslenkungstabelle veranlaßt. Nachdem die Tabellenberechnung abgeschlossen ist, müssen neue Distanzvektoren an die Nachbarn gesendet werden. Innerhalb der Methode SendVector werden die entsprechenden Vektoren für die Nachbarn zusammengestellt und mit Steuerinformationen für den Ausgangsdispatcher versehen. Anschließend werden sie an diesen gesendet. Der Dispatcher packt den Vektor wieder aus und bestimmt anhand der Steuerinformation die richtige Ausgangswarteschlange (vgl. Abschnitt 4.5.1). Abschließend wird der Vektor an die entsprechende Warteschlange gesendet.

Nachdem alle Knoten die bis dahin empfangenen Pakete bearbeitet und neue Protokollpakete erzeugt haben, müssen die in den Ausgangswarteschlangen wartenden Pakete über die Leitungen übertragen werden. Dazu sendet das Netzobjekt der Reihe nach die Meldung TransferPackets an alle Leitungen. Diese schließen daraufhin ihre internen Schalter, und die Pakete werden von den Ausgangswarteschlangen eines Knotens zum

Eingangsmultiplexer des nächsten Knotens übertragen. Dort wird die notwendige Steuerinformation hinzugefügt, damit später die Verkehrslenkung des Knotens den Sender des Pakets identifizieren kann (vgl. Abschnitt 4.5.1). Anschließend wird das Paket in die Eingangswarteschlange des Knotens übertragen. Dort wartet es solange, bis im nächsten Takt der Schalter des Knotens erneut geschlossen und das Paket bearbeitet wird.

Dieses Szenario wiederholt sich solange, bis die Simulation entweder durch die Ablaufsteuerung automatisch oder durch den Anwender über die interaktive Schnittstelle beendet wird.

## 4.6 Ein allgemeines Konzept zur Überwachung des Protokollablaufs

### 4.6.1 Motivation

Besondere Schwierigkeiten bei der Untersuchung von Verkehrslenkungsprotokollen macht der verteilte, asynchrone Ablauf des Protokolls. Dadurch sind Zusammenhänge schwerer erkennbar. In Abbildung 4.11 ist ein sehr einfaches Netz dargestellt, bei dem jede Leitung eine Länge von einer Einheit habe. Als Verkehrslenkungsalgorithmus werde der verteilte Bellman-Ford-Algorithmus angenommen. Im Inneren des Netzes sind die Längen der kürzesten Wege zum Knoten H im eingeschwungenen Zustand angegeben. Zum Zeitpunkt 0 falle die Leitung zwischen den Knoten A und H aus. Hat man keine Möglichkeit, den Ablauf des Protokolls zu verfolgen, stellt man am Ende der Simulation nur fest, daß jeder Knoten nach einiger Zeit neue Verkehrslenkungstabellen besitzt und daß viele Meldungen ausgetauscht wurden. Um eventuelle Fehler im Protokoll erkennen oder Optimierungen

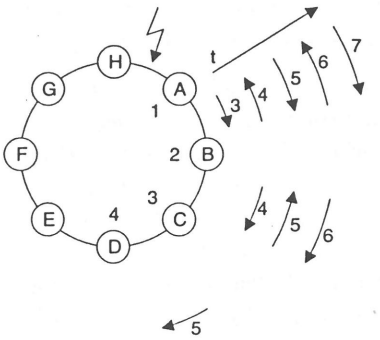



Bild 4.11: Einfaches Beispielnetz mit verteiltem Bellman-Ford-Algorithmus

vornehmen zu können, reicht dies nicht aus. Dazu ist ein genaues Nachvollziehen des Protokollablaufs und des Nachrichtenaustauschs notwendig.

In Abbildung 4.11 sind die Protokollpakete eingezeichnet, die sich auf die Entfernung zum Knoten H beziehen. Aus dem Nachrichtenaustausch erkennt man sehr gut, daß Pakete zwischen jeweils zwei Knoten pendeln. Die Entfernung wird schrittweise erhöht, bis der endgültige Wert erreicht ist. Im Prinzip wird hier wieder das „counting to infinity" Problem sichtbar, nur mit dem Unterschied, daß der Endwert nicht $\infty$ ist (vgl. Abbildung 3.4).

Werden größere Netze untersucht, können sehr schnell große Mengen an Überwachungsinformation anfallen. Hier müssen geeignete Mechanismen vorhanden sein, um die Informationen noch übersichtlich zu halten. Die Anforderungen, die ein Überwachungskonzept erfüllen muß, können wie folgt zusammengefaßt werden:

- Pakete können eine Spur hinterlassen, anhand der ihr Weg durch das Simulationsmodell nachvollziehbar wird.
- Jede Modellkomponente kann eine Meldung ausgeben, wenn sie Pakete empfängt, sendet oder auf andere Weise ihren internen Zustand ändert.
- Meldungen können je nach Wichtigkeit und Informationsgehalt in verschiedene, frei wählbare Kategorien eingeteilt werden.
- Um die Informationsflut zu reduzieren, muß der Anwender die Möglichkeit haben, eine Auswahl zu treffen, d.h. er muß genau festlegen können, welche Pakete eine Spur hinterlassen und welche Modellkomponenten wann Meldungen ausgeben. Es sollten auch Kombinationen aus beiden möglich sein, d.h. bestimmte Pakete hinterlassen nur an ganz speziellen Modellkomponenten eine Spur. Ferner sollte eine Auswahl der Meldungskategorien möglich sein, d.h. es werden grundsätzlich nur Meldungen ausgegeben, die zu einer bestimmten Kategorie gehören.
- Um die Übersichtlichkeit der Ausgabe zu steigern, sollten die Ausgaben nach verschiedenen Kriterien sortiert und auf unterschiedliche Dateien ausgegeben werden können. Es sollten mehrere Sortierkriterien gleichzeitig möglich sein, d.h. eine Meldung kann durchaus in mehreren Dateien erscheinen. Man könnte sich hier z.B. vorstellen, daß Meldungen, die vom Knoten A erzeugt werden, getrennt von den Meldungen des Knoten B aufgeführt werden. Um den zeitlichen Zusammenhang zu bewahren, werden alle Meldungen zusätzlich in einer dritten Datei gemeinsam aufgelistet. In einer vierten Datei stehen alle Meldungen, die Veränderungen in Verkehrslenkungstabellen betreffen.
- Die Steuerung der Ausgabe muß dynamisch zur Laufzeit möglich sein.

- Das Konzept sollte sich möglichst natürlich in die Simulationsumgebung integrieren, d.h. möglichst viele Gemeinsamkeiten mit normalen Ausgabeoperationen besitzen. Diese haben in der Sprache C++ die Form:

```
Ausgabestrom << "Text" << setw(20)
             << "weiterer Text" << endl;
```

  Die Ausgabe erfolgt mit Hilfe des Operators „<<“, wobei mehrere Ausgaben aneinandergekettet werden können. Formatieranweisungen (wie `setw(...)` bzw. `endl`) können zwischen dem eigentlichen Text stehen. Man spricht in diesem Zusammenhang von sogenannten Manipulatoren.

  Überträgt man diese Ausgabeform auf das Überwachungskonzept, ergibt sich, daß:

  - die Kettung von Ausgabetext (Meldungen) möglich sein muß und
  - die Steuerung der Ausgabe über Manipulatoren erfolgt.

### 4.6.2 Realisierung

Um die oben gestellten Anforderungen an die Flexibilität der Überwachung zu erfüllen, wurden folgende Grundgedanken umgesetzt: Der Anwender kodiert an bestimmte Stellen im Quelltext Ausgabeoperationen für seine Meldungen. Diese Stellen werden als Kontrollpunkte bezeichnet. Eine Meldung kann, aber muß nicht, ausgegeben werden, wenn während des Programmablaufs ein Kontrollpunkt passiert wird.

Zur besseren Strukturierung können gleichartige Kontrollpunkte zu Überwachungsebenen zusammengefaßt werden. Es ist z.B. sinnvoll, alle Kontrollpunkte zusammenzufassen, deren Meldungen Verkehrslenkungstabellen betreffen. Die Zuordnung von Kontrollpunkten zu Ebenen erfolgt vor dem Übersetzen eines Programms und kann damit während des Programmablaufs nicht mehr verändert werden. Jeder Ebene wird eine eindeutige Kennung zugeordnet. Ein Kontrollpunkt kann durchaus mehreren Ebenen angehören. Alle Ebenen eines Kontrollpunkts bilden die Menge $M_{Tracepoint}$.

Modellkomponenten können an der Überwachung mehrerer Ebenen teilnehmen. Eine Meldung wird nur dann ausgegeben, wenn der zugehörige Kontrollpunkt zu einer Ebene gehört, die von der Modellkomponente überwacht wird. Jede Komponente besitzt ein Feld, in dem alle Ebenen eingetragen sind, die von ihr überwacht werden sollen. Die Menge der Ebenen wird als $M_{Entity}$ bezeichnet.

Dasselbe Prinzip wird auch bei Paketen angewendet. Pakete hinterlassen nur eine Spur, wenn sie Kontrollpunkte bestimmter Überwachungsebenen passieren. Diese werden zur Menge $M_{Packet}$ zusammengefaßt, die jedes Paket in einem eigenen Feld mitführt. Wird während der Programmausführung ein Kontrollpunkt passiert, wird die Schnittmenge aus

$M_{Entity}$, $M_{Packet}$ und $M_{Tracepoint}$ gebildet. Ist die Schnittmenge nicht leer, wird die Meldung des Kontrollpunkts ausgegeben.

Dieses Prinzip wird in Abbildung 4.12 anhand eines Beispiels verdeutlicht. Dort sind zwei Knoten A und B dargestellt, die hierarchisch aufgebaut sind. Kontrollpunkte wurden

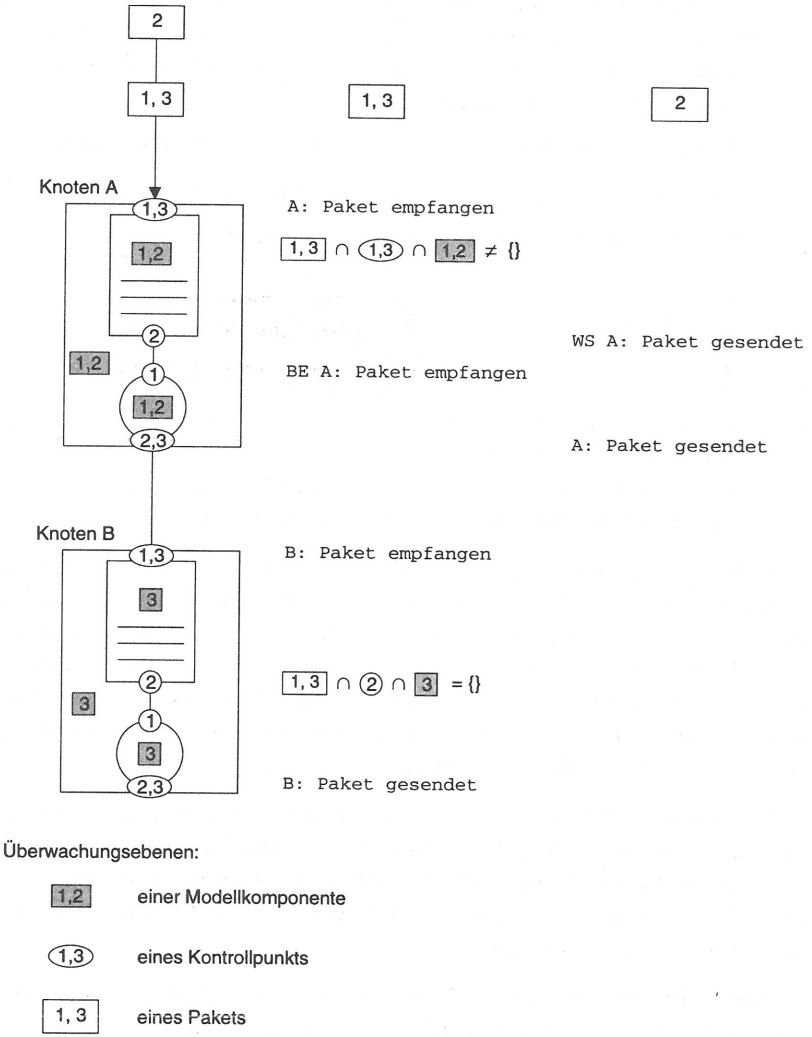



Bild 4.12: Beispiel für die Überwachung eines Protokolls

| Ebene | Kontrollpunkte am | Ausgabe |
|---|---|---|
| 1 | Eingang jeder Modellkomponente | `Name: Paket empfangen` |
| 2 | Ausgang jeder Modellkomponente | `Name: Paket gesendet` |
| 3 | Eingang eines Knotens | `Name: Paket empfangen` |
| | Ausgang eines Knotens | `Name: Paket gesendet` |

Tabelle 4.1: Überwachungsebenen des Beispiels aus Abbildung 4.12

an den Stellen im Code gesetzt, an denen Pakete von einer Modellkomponente empfangen und gesendet werden. In diesem Fall wurden drei Überwachungsebenen eingeführt. In Tabelle 4.1 sind die Ebenen und die zugehörigen Ausgabemeldungen aufgeführt.

Wie man aus Tabelle 4.1 erkennt, werden auf Ebene 3 nur die Kontrollpunkte am Eingang oder Ausgang eines Knotens zusammengefaßt. Der hierarchische Aufbau eines Knotens und die Kontrollpunkte der internen Modellkomponenten gehören nicht zur Ebene 3. Die Menge $M_{Entity}$ der verschiedenen Modellkomponenten sind in der Abbildung 4.12 ebenfalls angegeben. Die Spuren, die zwei Pakete mit verschiedenen $M_{Packet}$-Feldern beim Durchlaufen des Modells hinterlassen, sind neben dem Modell dargestellt.

Bis auf die Menge $M_{Tracepoint}$, die bei der Kodierung des Programms einmal festgelegt wird und danach nicht mehr geändert werden kann, ohne den Programmcode zu ändern, werden die beiden anderen Mengen zur Laufzeit verwaltet, d.h. zu jeder Menge können beliebig neue Ebenen hinzugefügt oder bestehende entfernt werden. Dadurch wird eine flexible, dynamische Steuerung der Überwachung zur Laufzeit ermöglicht.

Die Einsatzmöglichkeiten des Konzepts sind nicht nur auf den vorgestellten Protokollsimulator beschränkt. Auf Grund der hohen Flexibilität und der Möglichkeit, die Ausgabe zur Laufzeit steuern zu können, kann es auch für andere Softwareprojekte sehr wertvoll sein. Das Konzept ist vollständig in Klassen gekapselt und läßt sich so einfach integrieren.

## 4.7 Wiederverwendbare Entwurfsmuster

In diesem Abschnitt sollen einige Entwurfsmuster vorgestellt werden, die innerhalb der neu entwickelten Teile des Simulators eingesetzt wurden. Diese Muster haben allgemeinen Charakter und sind somit auch in anderen Anwendungen wiederverwendbar. Zu jedem Muster werden kurz das Entwurfsproblem, das das Muster löst, der Aufbau des Musters sowie ein Beispiel aus dem Kontext des Simulators erläutert.

### 4.7.1 Objektfabrik

Ein Vorteil der objektorientierten Programmierung liegt in der Unterstützung von Abstraktion und Polymorphie. Abstrakte Basisklassen werden dazu verwendet, die Gemeinsamkeiten mehrerer Klassen zusammenzufassen und eine gemeinsame Schnittstelle anzubieten. Konkrete Klassen, d.h. Klassen, von denen Objekte instanziiert werden, erhält man durch Ableiten von den abstrakten Basisklassen. Die Methoden der Schnittstelle werden erst in den abgeleiteten Klassen implementiert. Da Bibliotheken und Frameworks den genauen Typ eines Objekts und die Implementierung der Methoden nicht kennen müssen, verwenden sie meistens abstrakte Basisklassen für die Realisierung ihrer Aufgaben. Wollen Anwender die Dienste einer Bibliothek nutzen, leiten sie ihre anwendungsspezifischen Klassen von den abstrakten Basisklassen der Bibliothek ab.

Eine Bibliothek ist häufig auch für die Erzeugung der Objekte verantwortlich, die sie verwendet. Beim Entwurf der Bibliothek ist aber noch nicht bekannt, für welche Anwendungen sie verwendet werden soll und mit welchen anwendungsspezifischen Klassen sie zusammenarbeiten muß. Das Dilemma besteht darin, daß die Bibliothek Objekte von Klassen erzeugen muß, die sie nicht kennt. Sie kennt nur die abstrakten Basisklassen, von denen aber keine Objekte erzeugt werden können.

Als Beispiel für dieses Problem kann der Netzparser dienen, mit dem die Topologie eines Netzes eingelesen und dieses anschließend aufgebaut wird. Die Klasse TSubnetParser muß die Parameter von Knoten und Leitungen für ein Subnetz einlesen, entsprechende Objekte erzeugen und diese abschließend zu einen Netz der entsprechenden Topologie zusammenfügen, obwohl sie die exakten Klassen der Knoten bzw. Leitungen nicht kennt. Diese hängen von der jeweiligen Anwendung ab und werden erst zu einem späteren Zeitpunkt definiert.

Das Problem läßt sich durch die Einführung einer sogenannten Objektfabrik lösen. Durch dieses Muster wird das Wissen über die Klasse, die erzeugt werden soll, gekapselt und aus der Bibliothek ausgelagert. Die Grundstruktur des Musters ist in Abbildung 4.13 dargestellt. Es besteht aus zwei parallelen Klassenhierarchien, ObjectFactory und Product. Product ist eine abstrakte Basisklasse, die die gemeinsame Schnittstelle aller Objekte definiert, die in der Bibliothek verwendet werden. Die Bibliothek selbst arbeitet nur mit dieser Klasse. Von ihr werden die konkreten Anwendungsklassen, wie hier MyProduct, abgeleitet. Die Implementierung der Produktschnittstelle erfolgt in den abgeleiteten Klassen. Produktobjekte werden nicht direkt durch Kodierung eines entsprechenden Befehls in der Bibliothek erzeugt, sondern indirekt über Objektfabriken. Die Klasse ObjectFactory ist eine abstrakte Basisklasse für alle Objektfabriken. Sie kapselt die Erzeugung von Objekten in einer virtuellen Methode CreateProduct. Zu jeder Produktklasse gibt es eine korrespondierende Objektfabrik, die die CreateProduct-Methode überschreibt und ein Objekt der zugehörigen Produktklasse erzeugt.

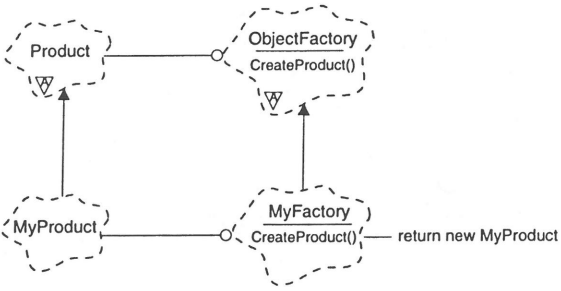



Bild 4.13: Struktur einer Objektfabrik

Die Integration neuer Anwendungsklassen in eine Bibliothek erfolgt nun einfach dadurch, daß der Anwender eine neue Objektfabrik für seine Anwendungsklasse bereitstellt und der Bibliothek anschließend mitteilt, von welcher Objektfabrik neue Objekte erzeugt werden sollen.

Für das Beispiel des Netzparsers stellen die Knoten bzw. Leitungen die Produkte und die Knoten- bzw. Leitungsparser die zugehörigen Objektfabriken dar. Neue Knoten bzw. Leitungsobjekte werden innerhalb des Subnetzparsers durch Aufruf der CreateNode bzw. CreateLink-Methoden eines Knoten- bzw. Leitungsparsers erzeugt. Alle Knoten- und Leitungsparser registrieren sich beim Subnetzparser. Die Auswahl, welche Objektfabrik die Objekte erzeugen soll, erfolgt mittels eines Schlüsselworts, das bei der Registrierung angegeben wird.

### 4.7.2 Prototyp

Das Muster Prototyp befaßt sich ebenfalls mit der Erzeugung neuer Objekte, deren exakte Klasse unbekannt ist. Allerdings existiert ein Prototyp des Objekts, der kopiert werden kann.

Bei der Verkehrslenkungsstrategie Flooding z.B. wird jedes ankommende Paket kopiert und auf allen abgehenden Leitungen ausgesendet. Der Kopiervorgang innerhalb des Flooding-Objekts muß dabei unabhängig vom Typ des empfangenen Pakets durchgeführt werden können. Ein Konstrukt, bei dem zuerst der Typ des Pakets abgefragt und dann mit Hilfe einer Verzweigung die richtige Kopie erzeugt wird, ist auf jeden Fall zu vermeiden. Solche Konstrukte sind äußerst unflexibel und änderungsunfreundlich. Für jeden neuen Pakettyp, der innerhalb des Netzes verschickt werden soll, muß der Code des Flooding-Objekts geändert und neu übersetzt werden.

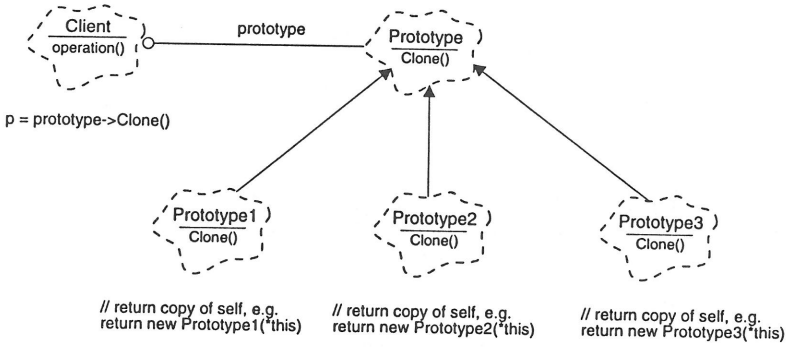



Bild 4.14: Struktur des Prototyp-Musters

Die Struktur des Prototyp-Musters ist in Abbildung 4.14 dargestellt. In der Basisklasse PROTOTYPE wird eine Schnittstelle definiert, mit deren Hilfe der Prototyp sich selbst kopiert (klont). Die Implementierung dieser Schnittstelle erfolgt in den jeweiligen Produktklassen, die von PROTOTYPE abgeleitet werden. Will ein anderes Objekt, der sogenannte Klient, eine Kopie des Prototyps erstellen, so fordert es den Prototyp auf, sich selbst zu klonen. Da sich die Produktobjekte polymorph verhalten, wird immer die eigene Kopierfunktion ausgeführt und nicht die einer eventuellen Basisklasse. Auf diese Weise wird sichergestellt, daß das kopierte Objekt immer den richtigen Typ besitzt und vollständig kopiert wird.

Das Muster verbirgt, ähnlich wie die Objektfabrik, den wirklichen Typ des Objekts, das kopiert werden soll. Es trägt somit zu einer Reduzierung der Anzahl der Namen bei, die ein Klient kennen muß. Außerdem ermöglicht es, daß Klienten mit anwendungsspezifischen Klassen zusammenarbeiten, ohne daß Änderungen im Code notwendig werden. Zusätzliche Flexibilität erhält man dadurch, daß ein System sogar zur Laufzeit um neue Produkte erweitert werden kann. Es ist nur dafür zu sorgen, daß ein entsprechender Prototyp des Produkts beim jeweiligen Klienten registriert wird. Dies kann dynamisch erfolgen.

Das beschriebene Muster wurde in dem Simulator bei allen Verkehrslenkungsmeldungen verwendet. Dazu wurde eine spezielle Klasse TROUTINGMESSAGE als Prototyp für alle weiteren Protokollmeldungsklassen eingeführt.

### 4.7.3 Globale Instanzen

Vielfach ist es für die Funktion einer Anwendung wichtig, daß von bestimmten Klassen genau eine Instanz innerhalb des Systems existiert, auf die von jeder Stelle des Programms über eine genau definierte Schnittstelle zugegriffen werden kann. Man kann sich hier z.B.

ein Managerobjekt vorstellen, das alle Ressourcen des Systems verwaltet und auf Anfrage zuteilt. Diese Verwaltung und Zuteilung der Ressourcen kann nur dann richtig funktionieren, wenn alle Programmteile den gleichen Ressourcenmanager verwenden. Würden mehrere Manager für dieselben Ressourcen existieren, könnte es zu Mehrfachvergaben kommen.

Durch eine globale Variable wird ein Objekt zwar global sichtbar, und es kann von überall auf sie zugegriffen werden, sie verhindert aber nicht, daß mehrere Objekte der zugehörigen Klasse instanziiert werden. Eine bessere Lösung erhält man, wenn die Klasse selbst dafür verantwortlich ist, daß nur eine Instanz von ihr existiert. Sie kann über die Zahl der Instanziierungen Buch führen und die Erzeugung neuer Objekte verhindern, falls bereits eine Instanz von ihr existiert. Zusätzlich kann sie einen einheitlichen Zugang zu dem globalen Objekt zur Verfügung stellen.

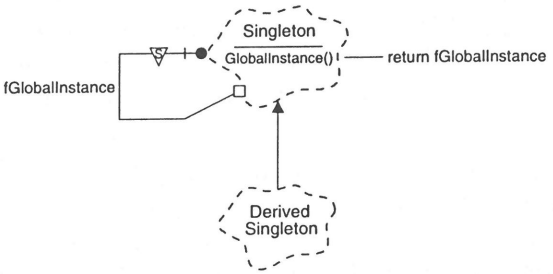



Bild 4.15: Struktur des Singleton-Musters

Die Grundstruktur des Musters ist in Abbildung 4.15 dargestellt. Die Klasse SINGLETON besitzt eine statische[1] Referenz (oder Zeiger) auf ein Objekt der eigenen Klasse. Dieses Objekt stellt die globale Instanz der Klasse dar. Das Objekt ist ein protected Feld der Klasse, so daß nur die Klasse selbst oder abgeleitete Klassen direkten Zugriff haben. Alle anderen Objekte erhalten einen einheitlichen Zugriff auf die globale Instanz durch die ebenfalls statische Methode GlobalInstance. Diese sorgt auch dafür, daß es genau eine Instanz der Klasse gibt. Beim ersten Aufruf erzeugt[2] sie die globale Instanz und liefert diese als Rückgabewert. Bei allen weiteren Aufrufen wird kein neues Objekt mehr erzeugt, sondern nur noch die globale Instanz zurückgeliefert.

[1]Statische Member (Felder oder Methoden) einer Klasse sind nicht in jeder Instanz dieser Klasse enthalten, sondern existieren genau einmal. In Smalltalk werden sie z.B. als Klassenvariablen bzw. Klassenmethoden bezeichnet.

[2]Die Erzeugung von Instanzen der Klasse von beliebiger Stelle im Programm kann dadurch verhindert werden, daß der Konstruktor, der jedesmal aufgerufen wird, wenn ein neues Objekt erzeugt wird, ebenfalls protected gemacht wird.

Oftmals soll die Funktion eines globalen Mangers nach einiger Zeit erweitert werden und der bisherige Mangager durch den neuen, erweiterten ersetzt werden. Dieses Erweitern der Funktionen und Ersetzen des Managers sollte ohne Änderung des bestehenden Codes erfolgen.

Mit dem vorgestellten Muster ist dies sehr leicht möglich. Die Erweiterung der Funktionen kann durch Ableiten neuer Klassen von SINGLETON erfolgen. Abgeleitete Klassen haben Zugriff auf das statische Feld, das auf die globale Instanz verweist. Sie können es so mit einem Objekt ihre eigenen Klasse initialisieren, wodurch die globale Instanz automatisch die erweiterte Funktionalität besitzt. Da der Zugriff auf das globale Objekt über eine genau definierte Schnittstelle erfolgt, muß der bestehende Code nicht verändert werden.

Die Vorteile dieses Musters liegen zum einen darin, daß die globale Instanz vollständig in der Singleton Klasse gekapselt ist. So kann sehr einfach kontrolliert werden, wann und wie auf das globale Objekt zugegriffen wird. Zusätzlich wird eine „Verschmutzung" des Namensraums durch unzählige globale Variablen vermieden, was vor allem für größere Softwareprojekte von großer Wichtigkeit ist. Derselbe Ansatz kann außerdem dazu verwendet werden, die Anzahl ($> 1$) der Objekte einer Klasse in einer Anwendung zu kontrollieren.

Dieses Muster wurde in dem vorgestellten Werkzeug überall da verwendet, wo globale Manager Ressourcen des Gesamtsystems verwalten. Dies ist z.B. bei dem Subnetzparser der Fall, der alle dem Werkzeug bekannten Knoten- und Leitungsparser verwaltet und das Einlesen der entsprechenden Parameter an sie delegiert. Ein zweites Beispiel ist eine Netzwerkklasse, die alle Knoten und Leitungen verwaltet und eine Umsetzung von benutzerfreundlichen Namen, wie Knoten „Stuttgart" oder Leitung „Stuttgart-Mannheim", zu den entsprechenden Objekten vornimmt.

# Kapitel 5

# Ein neues Verfahren zur adaptiven Verkehrslenkung in Paketvermittlungsnetzen

## 5.1 Anforderungen an eine moderne Verkehrslenkung

Ein wesentliches Kriterium für die Güte und damit für die Akzeptanz eines Datennetzes ist dessen Verfügbarkeit und Zuverlässigkeit. Fällt ein Netz häufig aus oder können Kommunikationsbeziehungen auf Grund von Netzfehlern nicht aufgebaut werden, wird dieses sich nur schwer bei den Teilnehmern durchsetzen. Eine hohe Ausfallsicherheit des Netzes wurde im Weitverkehrsbereich bisher meist durch eine hohe Ausfallsicherheit jedes einzelnen Knotens bzw. der Übertragungsabschnitte erreicht. Dazu ist viel Redundanz innerhalb der Knoten (siehe z.B. [97]) notwendig, die diese stark verteuert. Die Verkehrslenkung spielt eine wichtige Rolle in Bezug auf die Verfügbarkeit und Zuverlässigkeit eines Netzes. Eine intelligente Verkehrslenkung sollte Ausfälle einzelner Leitungen erkennen, selbständig einen neuen Weg berechnen und bestehende Verbindungen umlenken. Ein Ausfall bleibt so vor dem Teilnehmer verborgen. Auf diese Weise ist es möglich, den Aufwand innerhalb eines Knotens zu reduzieren, auch vereinzelte Ausfälle von Knoten oder Leitungen zu tolerieren und damit die Kosten pro Knoten zu senken, ohne daß dem Teilnehmer dadurch Nachteile entstehen.

Um dem ständig steigenden Bedarf an Datenkommunikation gerecht zu werden, werden nicht nur neue Netze in Betrieb genommen, sondern auch bestehende Datennetze weiter ausgebaut. Die Deutsche Telekom hat z.B. die Zahl der Vermittlungsknoten in ihrem Datex-P Netz von ursprünglich 18 auf inzwischen über 100 (Stand Anfang 1995) erhöht. Dieser Ausbau wird auch in Zukunft fortgesetzt werden. Um einen schrittweisen

Ausbau der Netze für die Betreiber so einfach wie möglich zu gestalten, sollte das Hinzufügen neuer Knoten oder Leitungen während des normalen Netzbetriebs möglich sein. Das bedeutet, daß die Verkehrslenkung neue Knoten und Leitungen selbständig erkennen und bei der Bestimmung der Wege berücksichtigen muß. Ein neu hinzugefügter Knoten sollte in der Lage sein, sich alle Informationen über das Netz, die er zur Bestimmung der Wege benötigt, selbständig zu beschaffen. Ein manuelles Konfigurieren eines neuen Knotens oder das Umkonfigurieren der bestehenden Knoten von Seiten eines Netzadministrators sollte nicht notwendig sein.

Weiter oben wurde bereits die Anforderung gestellt, daß ein Verkehrslenkungsverfahren auf Leitungsausfälle selbständig reagieren muß. Fällt eine Leitung aus, müssen die Pakete von Verbindungen, die diese Leitung benutzt haben, entweder verworfen oder solange im Netz zwischengespeichert werden, bis die Verbindungen über einen neuen Weg wieder aufgebaut sind. Verworfene Pakete werden vom Sender wiederholt, so daß das Netz auch in diesem Fall zusätzlich belastet wird. Die Reaktion auf einen Ausfall sollte deshalb so schnell wie möglich erfolgen.

Erkennt ein Knoten eine signifikante Änderung in der Topologie des Netzes oder der Länge einer Leitung, wird er dies in der Regel allen anderen Knoten mitteilen und eine Neuberechnung der Wege vornehmen. Die Zeit, bis alle Knoten diese neuen Daten erhalten haben und ebenfalls ihre Wege neu bestimmt haben, ist besonders kritisch. Während dieser Zeit ist das Netz anfällig für temporäre Schleifen (vgl. Kapitel 3 auf Seite 45). Diese Dauer sollte so kurz wie möglich sein.

Das Verfahren sollte flexibel genug sein, um auch für zukünftige Netze, die unter Umständen auf anderen Übermittlungsverfahren basieren, eingesetzt werden zu können.

Schließlich sollte ein Verfahren natürlich auch die jeweils „besten“ Wege für jedes Quelle-Ziel-Paar berechnen.

## 5.2 Ein adaptives Verfahren mit Alternativwegen

### 5.2.1 Grundlegender Aufbau

In Kapitel 3 wurden die verschiedenen Verkehrslenkungsverfahren mit ihren Vor- und Nachteilen vorgestellt. Zentrale Verfahren sind zwar in der Realisierung zum Teil einfacher, da hier das Problem der Konsistenz der Verkehrslenkungsdaten in verschiedenen Knoten nicht existiert. Auf der anderen Seite sind zentrale Verfahren, wie alle zentralen Einrichtungen, sehr anfällig gegenüber Fehlern oder Ausfällen. Auf die weiteren Nachteile der zentralen Verkehrslenkung wurde bereits in Kapitel 3 hingewiesen. Aus diesen Gründen

soll hier eine verteilte Lösung entworfen werden. Verteilte Lösungen bieten zusätzlich den Vorteil, daß sie wesentlich schneller auf Änderungen reagieren können als zentrale Lösungen. Haben Topologieänderungen nur lokale Auswirkungen, ist die Konvergenzzeit, d.h. die Zeitspanne, bis das Netz nach einer Änderung wieder in einem stabilen Zustand ist, bei einem verteilten Ansatz ebenfalls kleiner.

Das entwickelte Verfahren ist grundsätzlich sowohl für die verbindungsorientierte als auch die verbindungslose Kommunikation geeignet. Im folgenden wird der Einfachheit halber nur noch die verbindungsorientierte Variante betrachtet. Wenn nachfolgend vom Aufbau einer Verbindung gesprochen wird, dann geschieht dies, weil zu diesem Zeitpunkt die Auswahl eines Wegs stattfindet.

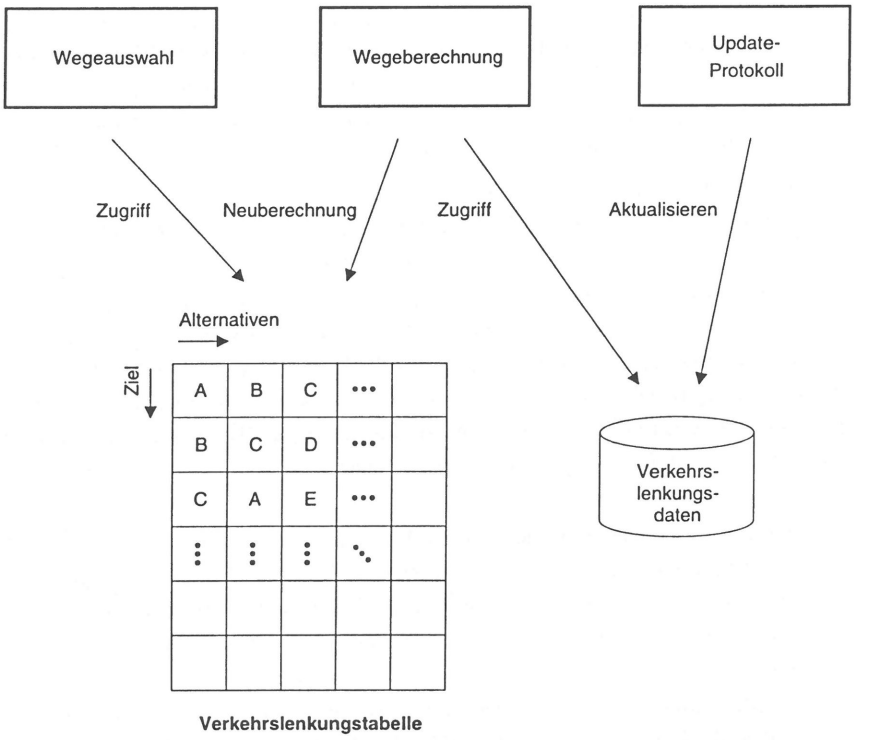



Bild 5.1: Komponenten des Verkehrslenkungsverfahrens

Die Komponenten des entwickelten Verkehrslenkungsverfahrens, die in jedem Netzknoten vorhanden sind, sind in Abbildung 5.1 dargestellt. Dies sind eine Verkehrslenkungstabelle mit mehreren Alternativen pro Zielknoten, eine Komponente zur Auswahl

eines Wegs aus den zur Verfügung stehenden Alternativen, eine Komponente zur Berechnung der Verkehrslenkungstabelle, eine Komponente, die spezielle Verkehrslenkungsdaten verwaltet, die für die Berechnung der Wege benötigt werden, und ein Protokoll, das diese Daten immer auf einem aktuellen Stand hält. Für die Realisierung der Komponenten stehen mehrere Alternativen zu Auswahl, die zum Teil direkt im Anschluß oder in den nachfolgenden Abschnitten diskutiert werden.

Da die Algorithmen zur Berechnung der Wege iterativ vorgehen, ist es nicht ratsam, sie bei jedem Verbindungsaufbau aufs neue aufzurufen und das Ergebnis der Berechnung abzuwarten. Es ist sinnvoller, wie in Abbildung 5.1 dargestellt, die Berechnung und die Auswahl eines Wegs in zwei Komponenten aufzutrennen und unabhängig voneinander durchzuführen. Die Ergebnisse, die man nach dem Aufruf eines Berechnungsalgorithmus erhält, werden in einer Verkehrslenkungstabelle abgespeichert. Beim Aufbau einer Verbindung wird dann nur noch auf diese Tabelle zugegriffen und die gespeicherten Informationen entsprechend ausgewertet. Da ein Zugriff auf die Tabelle wesentlich schneller ist als die Berechnung eines Wegs mit Hilfe eines iterativen Algorithmus, kann man auf diese Weise wesentlich kürzere Verbindungsaufbauzeiten erreichen.

Um möglichst schnell auf eine Änderung im Netz reagieren zu können, werden in der Verkehrslenkungstabelle mehrere Alternativwege bereitgestellt. Fällt der Erstweg aus oder ist er überlastet, kann sofort auf den Zweitweg ausgewichen werden.

Die Art und Weise, wie ein Verbindungsaufbau erfolgt, hat entscheidenden Einfluß auf den Inhalt der Verkehrslenkungstabelle und ihren Berechnungsalgorithmus. Wird source routing verwendet, muß für jeden möglichen Zielknoten ein vollständiger Weg in der Verkehrslenkungstabelle gespeichert werden. Alternativwege können nur beim jeweiligen Quellknoten berücksichtigt werden. Dieser hat dann in seiner Verkehrslenkungstabelle pro Zielknoten nicht nur einen vollständigen Weg gespeichert, sondern mehrere. Die Nachteile dieser Lösung sind offensichtlich:

- Für die Speicherung der Wege in den Verkehrslenkungstabellen wird wesentlich mehr Speicherplatz benötigt. Dies macht sich um so deutlicher bemerkbar, je größer die Netze und je länger die Wege damit in der Regel werden. Alternativwege können wegen des hohen Speicherbedarfs nur in sehr begrenztem Umfang berücksichtigt werden.
- Die Mitführung des vollständigen Wegs in jedem Verbindungsaufbaupaket bedeutet eine zusätzliche Belastung. Das Verhältnis zwischen Nutz- und Steuerinformation verschlechtert sich nochmals deutlich, wenn verbindungslose Kommunikation verwendet wird, denn dann muß jedes Datenpaket den vollständigen Weg mit sich führen.
- Da der Weg, den ein Paket durch das Netz nimmt, bereits im jeweiligen Quellknoten bestimmt wird, kann nur dort auf Topologieänderungen des Netzes reagiert werden.

Die Reaktionszeit ist somit länger, da die Knoten erst nach und nach von einer Änderung erfahren.

Wird eine Verbindung abschnittsweise aufgebaut, muß in jedem Knoten nur der jeweils nächste Wegabschnitt bestimmt werden. Die Verkehrslenkungstabelle eines Knotens enthält dann pro Zielknoten und pro Alternative nur den nächsten Nachbarn in Richtung auf das Ziel. Diese Variante bietet außer dem geringeren Speicherbedarf und dem geringeren Overhead in den Verbindungsaufbaupaketen zusätzlich den Vorteil, daß bei einem Ausfall einer Leitung alle Pakete sofort auf einen Alternativweg umgeleitet werden können. Es muß nicht gewartet werden, bis die jeweiligen Ursprungsknoten der Pakete den Ausfall gemeldet bekommen, wie dies beim source routing der Fall ist. Die Reaktionszeit ist deshalb wesentlich geringer. Aus diesen Gründen wird für die weiteren Untersuchungen kein source routing verwendet, sondern die Wegauswahl abschnittsweise durchgeführt.

In [16, 48, 109] wurde gezeigt, daß Shortest Path-Algorithmen im praktischen Einsatz einer optimalen Lösung sehr nahe kommen. Da Shortest Path-Algorithmen zudem für eine verteilte Berechnung wesentlich besser geeignet sind, wird für die Bestimmung der Verkehrslenkungstabelle ein Algorithmus dieser Klasse verwendet. Durch die Wahl einer entsprechenden Metrik zur Bewertung der Übertragungsabschnitte erhält man die notwendige Flexibilität, um das vorgeschlagene Verfahren in verschiedenen Netzen einsetzen zu können.

Wie oben bereits erwähnt wurde, erfolgt die Bestimmung der Wege unabhängig von der Auswahl eines Wegs beim Verbindungsaufbau. Da moderne Vermittlungsknoten mehrere dedizierte Prozessoren besitzen, kann diese Bestimmung als eine Art Hintergrundaufgabe erfolgen. In der Tabelle werden für jeden Zielknoten mehrere Alternativen nach ihrer Länge sortiert gespeichert. Der erste Eintrag entspricht dem kürzesten Weg, der zweite Eintrag dem zweitkürzesten usw. Für die Bestimmung der Einträge kann sowohl ein Link-State- als auch ein Distance-Vector-Algorithmus eingesetzt werden. Einzige Bedingung ist, daß der Algorithmus in der Lage ist, mehrere Alternativen nach ihrer Länge sortiert zu berechnen. Das Bereitstellen von Alternativen, auf die im Bedarfsfall sofort ausgewichen werden kann, ist in der Praxis nur sinnvoll, wenn diese Alternativen auch schleifenfrei sind. Diese eigentlich trivial lautende Forderung ist die in der Praxis am schwierigsten zu erfüllende, da ein Ausfall nicht sofort im gesamten Netz bekannt ist. Auf dieses Problem wird später noch genauer eingegangen.

Für die Bestimmung der Wege benötigen die Algorithmen bestimmte Informationen über die Topologie des Netzes oder die Länge einzelner Wegabschnitte. Bei einem Link-State-Algorithmus z.B. muß jeder Knoten die komplette Topologie des Netzes und die Länge aller Leitungen kennen (vgl. Kapitel 3.3.2). Bei einem Distance-Vector-Algorithmus sind darunter die Entfernungstabellen zu verstehen, die ein Knoten von seinen Nachbarn erhält. Die letzte Komponente in Abbildung 5.1, das Update-Protokoll, sorgt dafür, daß

diese Informationen immer auf einem aktuellen Stand sind. Dies geschieht durch besondere Protokolle, die den Informationsaustausch zwischen den Knoten regeln. Sie sind im allgemeinen an den jeweiligen Shortest Path-Algorithmus angepaßt.

### 5.2.2 Ungelöste Probleme

Im Zusammenhang mit dem oben beschriebenen Aufbau des Verfahrens ergeben sich im wesentlichen zwei Probleme:

- Es muß ein Shortest Path-Algorithmus gefunden werden, der mehrere Alternativen berechnen kann.
- Wird auf eine Alternative ausgewichen, darf dadurch keine Schleife entstehen.

Shortest Path-Algorithmen, die mehrere Alternativen bestimmen können, werden auch als $k$-Shortest Path-Algorithmen bezeichnet, wobei $k$ die Anzahl der Wege angibt. In der Literatur sind einige Beiträge zum Thema $k$-Shortest Path-Algorithmen zu finden. Die meisten dort vorgestellten Algorithmen bestimmen jedoch Wege, die sich in mindestens einem, aber nicht unbedingt im ersten Übertragungsabschnitt unterscheiden [39, 108, 135, 157, 161, 166]. Um für das oben beschriebene Verfahren eingesetzt werden zu können, müssen sich die Alternativen aber im ersten Übertragungsabschnitt unterscheiden. Zu diesem speziellen Problem ist bisher nur wenig bekannt [147].

Bei Link-State-Protokollen, bei denen jeder Knoten die gesamte Topologie des Netzes kennt, kann ein normaler Shortest Path-Algorithmus (z.B. der Dijkstra-Algorithmus) mehrfach angewendet werden. Nach jeder Berechnung wird die erste Leitung des kürzesten Wegs aus der Topologietabelle entfernt. Die Berechnung wird dann mit der neuen Topologie wiederholt. Dies wird genau $k$-mal durchgeführt. Grundsätzlich lassen sich so zwar die $k$ kürzesten Wege berechnen, das Verfahren ist aber nicht sehr effizient. In [147] wurde daher ein Algorithmus vorgestellt, der das Problem besser löst.

Der Berechnung mehrerer Alternativen mit Hilfe von Distance-Vector-Algorithmen wurde in der Literatur bisher nur wenig Aufmerksamkeit gewidmet. Das Problem liegt hier in der Schleifenfreiheit der Alternativwege. Distance-Vector-Algorithmen basieren auf Gleichung 3.2:

$$D_{ik}^{(h+1)} = \min_j \left[ D_{ij}^{(h)} + d_{jk} \right] \qquad \forall\, k \neq i.$$

Da ein Weg ohne Schleife immer kürzer ist als ein Weg mit Schleife, wird durch die Minimierung garantiert, daß die kürzesten Wege gleichzeitig schleifenfrei sind. Der nächstlängere Weg entsprechend Gleichung 3.2 muß nicht zwangsläufig ein gültiger Weg sein, wie das Beispiel in Abbildung 5.2 zeigt. Knoten A möchte die Wege zu Knoten F berechnen. Die relevanten Teile der Distanzvektoren, die A von seinen Nachbarn erhält, sind in der

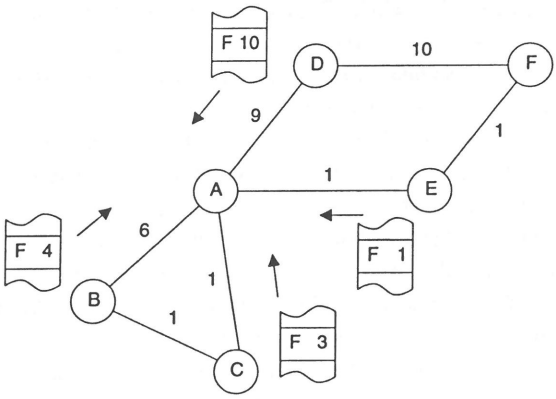



Bild 5.2: Berechnung von Alternativen mit Distance-Vector-Algorithmen

Abbildung ebenfalls eingezeichnet. Die Minimierung nach Gleichung 3.2 führt auf den kürzesten Weg (A,E,F) der Länge $D_{AF,E} = 2$. Der nächst größere Wert wäre $D_{AF,C} = 4$ für den Weg über Nachbar C. Wie man aus der Abbildung leicht sieht, ist dies keine gültige Alternative, da der Weg von C nach F bereits über A führt. Das gleiche gilt für den Weg über Nachbar B. Erst der letzte Wert $D_{AF,D} = 19$ für den Weg über D ergibt eine gültige Alternative.

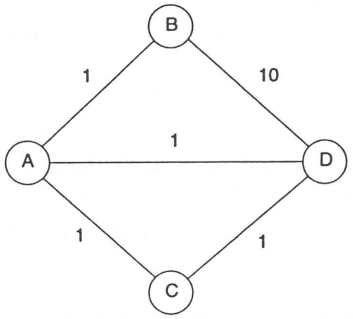



Tabelle Knoten A:

| Ziel | Nachbar | Wege |
|---|---|---|
| D | D C B | A,D A,C,D A,B,D |

Tabelle Knoten B:

| Ziel | Nachbar | Wege |
|---|---|---|
| D | A D | B,A,D B,D |

Bild 5.3: Beispielnetz mit mehreren Alternativwegen

Das zweite Problem ergibt sich bei der Auswahl eines Wegs beim Verbindungsaufbau. In Abbildung 5.3 ist ein einfaches Beispielnetz dargestellt. Betrachtet werden soll ein Verbindungsaufbau zwischen den Knoten A und D. Für die Knoten A und B sind entsprechende Ausschnitte aus den Verkehrslenkungstabellen in der Abbildung dargestellt. Zu jedem Eintrag in der Verkehrslenkungstabelle ist der zugehörige schleifenfreie Weg ebenfalls angegeben. Solange Knoten A nur seinen Erstweg, der in diesem Fall ein Direktweg ist, zum Aufbau von Verbindungen nach D benutzt, entstehen keine Probleme. Weicht A allerdings auf seine erste Alternative über Nachbar B aus, entsteht eine Schleife. Normalerweise benutzt B für den Aufbau von Verbindungen seinen Erstweg. Dieser führt in diesem Fall über A. In diesem einfachen Beispiel läßt sich das Problem noch sehr einfach lösen. Es muß nur die Bedingung eingeführt werden, daß ein Weg nur dann benutzt werden darf, wenn dadurch der Aufbauwunsch nicht zu dem Knoten zurückgesendet werden muß, von dem er erhalten wurde. Schwieriger wird es allerdings, wenn eine Schleife aus mehr als nur zwei Knoten besteht, denn dann ist sie nicht sofort erkennbar.

### 5.2.3 Ähnliche Ansätze

In [138] wurde ein ähnlich aufgebautes Verfahren wie das in Abschnitt 5.2.1 vorgestellte für High-Speed-Netze vorgeschlagen. Es benutzt einen speziellen Algorithmus [137], bei dem die berechneten Alternativen keine gemeinsamen Knoten besitzen. Die Bestimmung der Alternativen erfolgt durch einen intensiven Meldungsaustausch zwischen den Knoten. Eine wesentliche Beschränkung des Verfahrens liegt darin, daß sich die Topologie des Netzes während des Meldungsaustauschs nicht ändern darf. Um Schleifen zu verhindern, darf nur einmal, und zwar beim Quellknoten, auf einen Alternativweg ausgewichen werden. Damit geht der Vorteil gegenüber dem source routing verloren.

In [153] wird ebenfalls ein Verfahren mit Alternativwegen vorgeschlagen. Allerdings wird für jeden Zielknoten nur jeweils ein Ausweichweg bereitgestellt, der kurzzeitig benutzt wird, wenn der Erstweg überlastet ist. Dadurch wird versucht, die „Schwingneigung" von Shortest Path-Algorithmen (vgl. Abschnitt 3.3.3) zu reduzieren. Jeder Knoten berechnet zwei Tabellen. Die erste ist eine normale Verkehrslenkungstabelle, die einen Erstweg und maximal eine Alternative enthält. Der Erstweg wird durch einen einfachen Link-State-Algorithmus bestimmt. Die zweite Tabelle enthält spezielle Einträge für sogenannte Ausweichpfade. Sie wird durch eine eigenes Protokoll erstellt, das zwischen den Knoten abgewickelt wird.

Um die Alternativen zu bestimmen und Schleifen beim Ausweichen auf diese zu vermeiden, berechnet jeder Knoten einen sogenannten Verkehrslenkungsbaum. Das ist ein Baum, an dessen Wurzel der berechnende Knoten selbst steht. Jeder andere Netzknoten wird entsprechend des berechneten Wegs von der Wurzel zu ihm an den Baum angehängt.

Zusätzlich zu seinem eigenen Verkehrslenkungsbaum berechnet jeder Knoten auch die Bäume seiner Nachbarn. Mit deren Hilfe kann er entscheiden, ob eine Schleife entsteht, wenn ein Paket zu einem bestimmten Nachbarn gesendet wird, oder nicht. Entsteht keine Schleife, ist dieser Nachbar eine gültige Alternative und kann ganz normal verwendet werden. Würde jedoch eine Schleife entstehen, wird das Paket speziell markiert. Diese Markierung führt dazu, daß in den nachfolgenden Knoten nicht die Erstwege, sondern die Ausweichpfade der zweiten Tabelle benutzt werden.

Die Nachteile des Verfahrens bestehen darin, daß jeder Knoten zusätzlich zu den eigenen Wegen auch die Wege seiner Nachbarn berechnen muß, um die jeweiligen Verkehrslenkungsbäume aufzubauen. Das Protokoll zur Erstellung der Ausweichpfade bedeutet eine zusätzliche Belastung. Es ist nur ein Alternativweg erlaubt und Pakete können auch höchstens einmal über eine Alternative gelenkt werden. Insgesamt ist das Verfahren für eine statische Umgebung ausgelegt, bei der weder schnell auf Ausfälle reagiert, noch Wege häufig neuberechnet werden müssen.

## 5.3 Realisierung mit Link-State-Algorithmen

In diesem Abschnitt soll untersucht werden, inwieweit die Komponente für die Wegeberechnung mit Hilfe von Link-State-Algorithmen realisiert werden kann.

Ist die Topologie des Netzes in jedem Knoten bekannt, kann z.B. der Algorithmus aus [147] verwendet werden, um eine Verkehrslenkungstabelle mit mehreren sortierten Alternativen zu berechnen. Das zweite Problem aus Abschnitt 5.2.2 ist bei Link-State-Algorithmen schwerer zu lösen. Will man source routing beim Ausweichen auf eine Alternative vermeiden, ist der Austausch zusätzlicher Informationen zwischen den Knoten notwendig. In [153] ist dazu schon ein aufwendiges Protokoll notwendig, obwohl hier nur eine Alternative berechnet wird.

Zu den bereits erwähnten Problemen kommt bei Link-State-Algorithmen ein drittes hinzu. Da die Berechnung der Wege auf der Kenntnis der Topologie des Netzes und der Längen der Leitungen beruht, müssen die Topologietabellen in allen Knoten identisch sein. Ist dies nicht der Fall, können beim Verbindungsaufbau Schleifen entstehen. Auch nach einem Ausfall müssen die Tabellen aller Knoten so schnell wie möglich abgeglichen werden. Für den Abgleich ist ein Broadcast-Verfahren zu wählen.

Die geringste Belastung des Netzes durch Broadcast-Meldungen entsteht, wenn die Meldungen entlang eines *minimum spanning trees* gesendet werden. Dies ist ein Baum, der jeden Netzknoten genau einmal enthält und an dessen Wurzel der sendende Knoten steht. Die Knoten sind innerhalb des Baums so angeordnet, daß die Verbindung zwischen

ihnen und der Wurzel der kürzeste Weg (shortest path) zwischen beiden ist. Der Nachteil dieses Broadcast-Verfahrens liegt darin, daß es sehr empfindlich gegenüber weiteren Ausfällen ist. Fällt während eines Broadcasts eine zweite Leitung aus, die Teil des spanning trees ist, erreicht die erste Meldung einen Teil des Netzes niemals. Muß mit häufigen Änderungen der Netztopologie gerechnet werden, führt dies zu einem „Auseinanderlaufen" der Topologietabellen und folglich zu falschen Wegberechnungen.

Die sicherste Methode, Topologiedaten im Netz zu verteilen, bietet das Flooding. Dadurch wird garantiert, daß eine Meldung jeden Knoten erreicht, solange mindestens noch ein Weg zu ihm besteht. Diese Sicherheit wird allerdings mit einer höheren Belastung des Netzes erkauft.

Ein Kompromiß zwischen beiden Verfahren besteht darin, den spanning tree gezielt um einzelne Leitungen zu erweitern. Bei einem allgemeinen Netz ist die Bestimmung dieser Leitungen allerdings schwierig. Da die Topologietabellen der neuralgische Punkt bei Link-State-Algorithmen sind, ist von Kompromissen an dieser Stelle abzuraten.

Meldungen, die Topologiedaten enthalten, müssen zusätzlich mit einer Folgenummer versehen werden. Wird dies nicht getan, kann es in bestimmten Situationen zu falschen Einträgen in den Topologietabellen mancher Knoten kommen. Abbildung 5.4 verdeutlicht dies.

Zum Zeitpunkt $t = 0$ ändert sich die Länge der Leitung zwischen den Knoten A und B. Knoten B sendet eine entsprechende Meldung an seine Nachbarn C und D. Knoten D erhält diese zum Zeitpunkt $t = 2$ und trägt die neue Länge in seine Tabelle ein. Zum Zeitpunkt $t = 4$ ändert sich die Länge der Leitung erneut. Auch jetzt sendet Knoten B eine entsprechende Meldung, die zum Zeitpunkt $t = 6$ beim Knoten D ankommt. Auf Grund der längeren Laufzeit zwischen den Knoten B und C sind dort noch beide Pakete unterwegs. Zum Zeitpunkt $t = 13$ erreicht das inzwischen veraltete Paket mit der ersten Längenänderung den Knoten D. Ohne Folgenummer kann dieser nicht entscheiden, ob es sich um ein altes Paket oder eine erneute Änderung handelt. Folglich trägt er die veraltete Information in seine Tabelle ein. Kurz darauf fällt die Leitung zwischen C und D aus. Dieser Ausfall hat zur Folge, daß das letzte Paket mit der korrekten Längenangabe für die Leitung A–B verlorengeht, und die Tabelle von D einen falschen Eintrag enthält.

Wird eine ausgefallene Leitung zwischen zwei Knoten wieder in Betrieb genommen, müssen beide Knoten ihre Topologietabellen abgleichen. Um entscheiden zu können, in welcher Tabelle die aktuelleren Daten enthalten sind, sind ebenfalls Folgenummern erforderlich.

Um solchen Effekten entgegenzuwirken und die Sicherheit weiter zu erhöhen, sollten die Topologietabellen der Knoten in regelmäßigen Abständen miteinander verglichen und eventuelle Unterschiede beseitigt werden. Dies erhöht allerdings die Belastung des Netzes

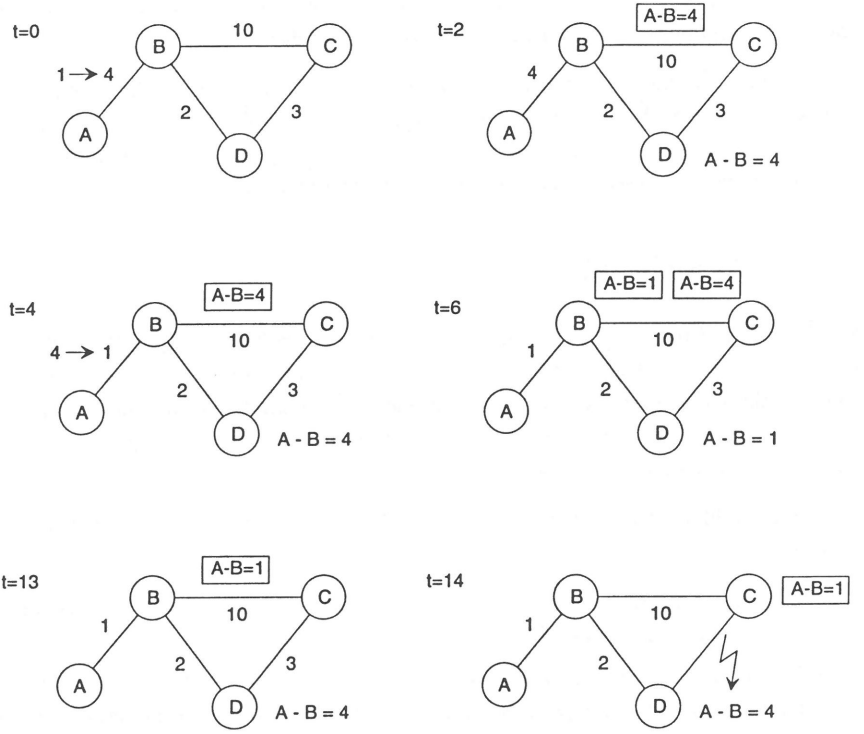



Bild 5.4: Beispiel für Notwendigkeit von Folgenummern

weiter. In welchen Abständen dies geschehen sollte, hängt vom jeweiligen Netz ab. Ist mit häufigen Topologieänderungen zu rechnen, sind die Abstände kleiner zu wählen als bei einem relativ stabilen Netz.

## 5.4 Entwurf eines neuen Protokolls

Im vorangegangenen Abschnitt wurden Link-State-Algorithmen näher untersucht. Damit die berechneten Alternativen im Bedarfsfall benutzt werden können, müßte auch dort ein eigenes Protokoll zum Informationsaustausch zwischen den Knoten entwickelt werden. Bei Distance-Vector-Algorithmen tauschen die Knoten bereits Informationen untereinander aus. Es stellt sich die Frage, ob mit Algorithmen dieser Klasse nicht auch die Berechnung mehrerer Alternativen möglich ist. Der Berechnung mehrerer Alternativen mit Hilfe eines Distance-Vector-Algorithmus wurde bisher in der Literatur wenig Aufmerksamkeit

gewidmet. In diesem Abschnitt wird ein neues Distance-Vector-Protokoll entworfen und schrittweise optimiert, das versucht, diese Lücke zu schließen.

Das neue Protokoll ist für den Einsatz im Zusammenhang mit der Architektur aus Abbildung 5.1 gedacht. Dafür ist eine Abstimmung der drei Komponenten Wegeauswahl, Berechnung der Alternativen und Update-Protokoll notwendig. In den folgenden Abschnitten wird auf diese Komponenten einzeln eingegangen.

### 5.4.1 Wegeauswahl

Ein wichtiges Kriterium für die Güte eines Netzes ist die Verbindungsaufbauzeit. Das vorgestelle Verfahren trägt dem Rechnung, indem es die Berechnung der Wege von der Auswahl eines konkreten Wegs beim Aufbau einer Verbindung trennt und im Hintergrund durchführt. Aus diesem Grund sollen auch die Aktionen, die bei einem Verbindungsaufbau durchzuführen sind, so einfach wie möglich sein.

Die Auswahl eines Wegs beim Verbindungsaufbau wird nach folgendem Prinzip durchgeführt:

- Jeder Knoten bestimmt selbst den nächsten Übertragungsabschnitt, d.h. es findet kein source routing statt.
- Der nächste Übertragungsabschnitt wird durch Lesen des jeweils ersten Eintrags (Erstweg) für den entsprechenden Zielknoten in der Verkehrslenkungstabelle bestimmt.
- Kann der Erstweg nicht benutzt werden, weil die Leitung z.B. gerade ausgefallen ist und die Verkehrslenkungstabelle noch nicht aktualisiert wurde, wird auf den ersten Alternativweg ausgewichen. Kann dieser ebenfalls nicht benutzt werden, wird die nächste Alternative genommen, usw. Wird auf diese Weise kein Weg gefunden, wird die Verbindung abgelehnt.
- Die Bildung von Schleifen wird vermieden. Würde die Wahl eines Wegs (Erstweg oder Alternative) zu einer Schleife führen, wird die Verbindung nicht über diesen Weg aufgebaut. Anstatt dessen wird die nächste Alternative in Betracht gezogen.

Zur Erkennung von Schleifen reichen die Informationen in der Verkehrslenkungstabelle alleine nicht aus. Anhand dieser Daten können nur unmittelbare Schleifen vermieden werden, d.h. es kann verhindert werden, daß ein Verbindungsaufbauwunsch zu dem Knoten zurückgeschickt wird, von dem er empfangen wurde. Um Schleifen zu vermeiden, die aus mehr als zwei Knoten bestehen, muß man wissen, über welchen Weg ein Nachbarknoten die Verbindung weiter aufbauen will. Auf diesen Punkt wird in Abschnitt 5.4.2 und 5.4.3 näher eingegangen.

### 5.4.2 Ein einfaches Update-Protokoll

In diesem Abschnitt wird ein einfaches Protokoll vorgestellt, mit dem die Knoten ihre Distanzvektoren untereinander austauschen. Diese erste Version des Protokolls wird im Abschnitt 5.5 optimiert. Die wesentlichen Eigenschaften des Protokolls sind:

- Die Distanzvektoren, die zu einem bestimmten Nachbarn gesendet werden, werden für diesen individuell berechnet.
- Ein Distanzvektor wird nur dann zu einem Nachbarknoten gesendet, wenn er sich vom zuletzt gesendeten unterscheidet.
- Bei der Berechnung der Distanzvektoren wird die Auswahlstrategie des Erst- bzw. Alternativwegs (vgl. Abschnitt 5.4.1) berücksichtigt.
- Zusätzlich zur Entfernungsinformation enthalten die Distanzvektoren für jeden Zielknoten Daten über den zugehörigen Weg.
- Die Distanzvektoren werden immer dann neu berechnet, wenn:
  - sich die Entfernung eines Knotens zu einem seiner Nachbarn geändert hat,
  - ein Knoten eine Meldung bzw. mehrere Meldungen mit neuen Distanzvektoren von seinen Nachbarn erhalten hat.

#### 5.4.2.1 Berechnung der Distanzvektoren

##### 5.4.2.1.1 Struktur eines Distanzvektors

Analog zum ursprünglichen Distance-Vector-Protokoll tauschen die Knoten untereinander Meldungen mit sogenannten Distanzvektoren aus. Diese sind aber gegenüber der ursprünglichen Form etwas modifiziert. Der grundsätzliche Aufbau des neuen Distanzvektors ist in Abbildung 5.5 dargestellt.

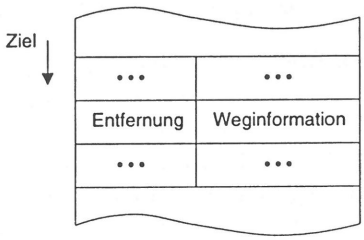



Bild 5.5: Aufbau des modifizierten Distanzvektors

Der neue Distanzvektor besitzt für jeden möglichen Zielknoten des Netzes einen Eintrag, der aus der Enfernung zum Ziel und der zugehörigen Weginformation besteht. Im Grunde genommen gehören zu der Weginformation alle Knoten des Wegs, einschließlich der Quell- und Zielknoten. Um die Größe eines Vektors etwas zu reduzieren, können bei den Einträgen jeweils der erste und letzte Knoten entfallen, da diese Information im Prinzip redundant vorhanden ist. Der erste Knoten jedes Wegs ist immer der Knoten, der den Distanzvektor versendet. Der letzte Knoten eines Wegs wird durch die entsprechende Zeile innerhalb des Vektors festgelegt. Es müssen also nur die Transferknoten eines Wegs übermittelt werden. Insbesondere bei großen Netzen, bei denen die Wege lang sind, können die Distanzvektoren sehr groß werden. Dieses Problem wird in Abschnitt 5.5.2 näher betrachtet.

Durch die zusätzliche Weginformation in den Distanzvektoren kennt ein Knoten den Weg, den ein Paket nehmen würde, wenn es zu einem bestimmten Nachbarn gesendet wird. Beim Verbindungsaufbau kann jeder Knoten anhand dieser Weginformation prüfen, ob er selbst bereits Teil des Wegs von diesem Nachbarn zum Ziel ist. Ist dies nicht der Fall, entsteht keine Schleife und der Verbindungsaufbauwunsch kann an den Nachbarn weitergeleitet werden.

#### 5.4.2.1.2 Individuelle Berechnung

Die Berechnung eines Distanzvektors erfolgt für jeden Nachbarknoten individuell. Dies erhöht zwar den Rechenaufwand etwas, ist aber zur frühzeitigen Vermeidung von Schleifen und zur Berechnung der Alternativwege unumgänglich. Abbildung 5.6 verdeutlicht dies. Der Erstweg von Knoten A nach C ist der Direktweg. Der Zweitweg führt über Knoten B und besitzt die Länge $D_{AC,B} = 11$. Knoten A kann diesen Weg nur dann berechnen, wenn er von Knoten B die Entfernung $d_{BC} = 10$ mitgeteilt bekommt. Wie man aus der Abbildung ebenfalls erkennt, stellt sich für Knoten D die Situation anders dar. Sein Weg zu Knoten C lautet (D,B,A,C) und besitzt die Länge $D_{DC} = 3$. Damit D diesen Weg korrekt berechnen kann, muß er von B die Entfernung $D_{BC} = 2$ mitgeteilt bekommen.

Der Unterschied zwischen beiden Fällen des obigen Beispiels liegt darin, daß B alle Verbindungen von D nach C über seinen eigenen Erstweg lenken kann. Würde B dies mit einer Verbindung von A nach C tun, entstünde eine Schleife, da der Erstweg von B über A verläuft. Wie in Abschnitt 5.4.1 bereits erwähnt wurde, erkennt ein Knoten solche Schleifen und benutzt automatisch die nächste gültige Alternative. Dies ist in diesem Fall der Direktweg (B,C) mit der Länge $d_{BC} = 10$. Die Distanz zu einem Ziel, die ein Knoten an seine Nachbarn meldet, muß folglich mit der Wegeauswahlstrategie beim Verbindungsaufbau in Einklang gebracht werden.

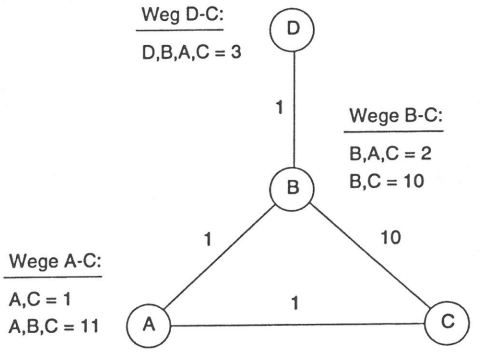



Bild 5.6: Notwendigkeit der individuellen Berechnung von Distanzvektoren

Bei der Berechnung der Distanzvektoren muß ein Knoten demnach für jedes Element des Vektors alle seine Wege (Erstweg und alle Alternativen) zum zugehörigen Zielknoten der Reihe nach auf Schleifen prüfen. Er wählt dann denjenigen aus, der am kürzesten und schleifenfrei ist. Die Schleifenprüfung erfolgt völlig analog zu der Prüfung beim Verbindungsaufbau. Anhand der Weginformation, die in den Distanzvektoren enthalten sind, kann ein Knoten feststellen, ob ein Nachbar bereits Teil eines Wegs ist. Kann kein schleifenfreier Weg gefunden werden, wird die Entfernung $\infty$ in das entsprechende Feld des Distanzvektors eingetragen.

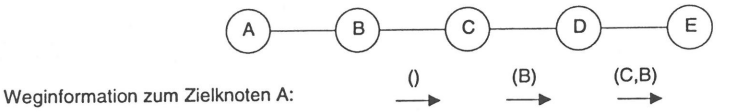



Bild 5.7: Beispiel für die Berechnung der Weginformation

Außer der Distanz zu einem Zielknoten übermittelt ein Knoten $i$ zusätzlich die Nummern der Transferknoten des entsprechenden Wegs. Diese Weginformation kann nach folgendem Schema ermittelt werden:

| Bedingung | Weginformation |
|---|---|
| $n = z$ | leer |
| $n \neq z$ | füge $n$ zu der Weginformation hinzu, die in Knoten $i$ von $n$ empfangen wurde |

Dabei bezeichnet $n$ den jeweils nächsten Knoten in Richtung auf das Ziel $z$. Der Nachbar $n$ kann der Verkehrslenkungstabelle entnommen werden. Die Weginformation baut sich so schrittweise vom Zielknoten aus auf. Dies ist in Abbildung 5.7 nochmals darge-

stellt. Der gesamte Algorithmus zur Berechnung der Distanzvektoren ist in Abbildung 5.8 dargestellt.

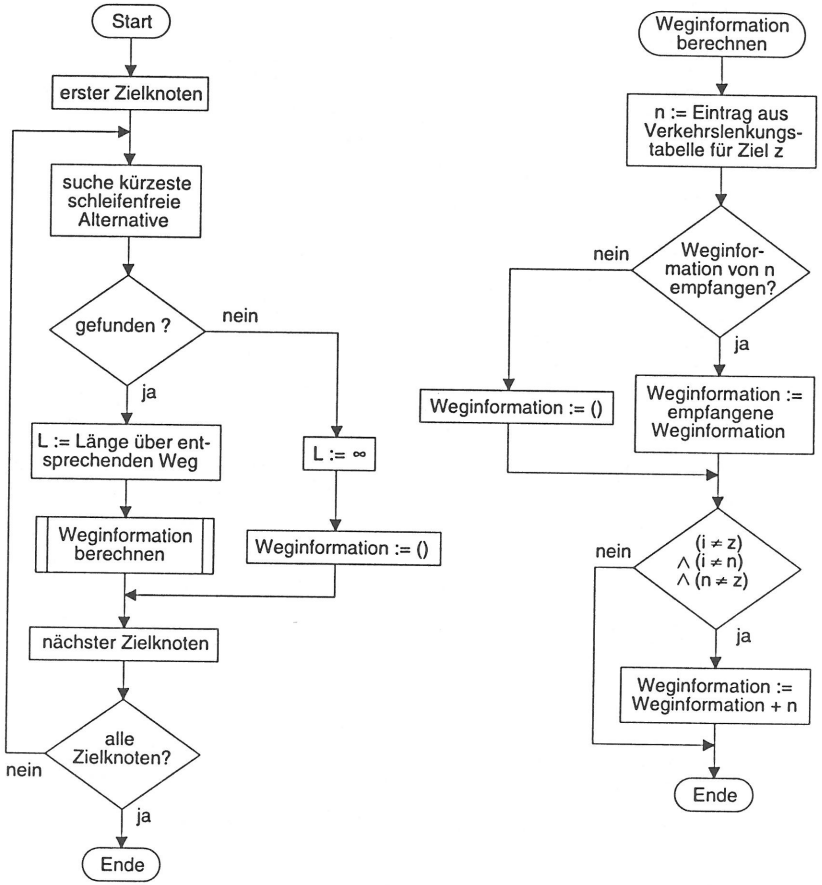



Bild 5.8: Algorithmus zur Berechnung eines Distanzvektors in Knoten $i$

#### 5.4.2.1.3 Berechnungszeitpunkt

In der Regel empfiehlt es sich, daß die Neuberechnung und das Versenden der Distanzvektoren nicht sofort nach dem Empfang einer einzelnen Meldung von einem Nachbarn durchgeführt wird. Dadurch könnte zwar theoretisch die kürzeste Reaktionszeit auf eine Topologieänderung erreicht werden, es würden aber unnötig viele Meldungen ausge-

tauscht. Hat eine Änderung der Topologie stattgefunden, ist es sehr wahrscheinlich, daß ein Knoten von mehreren Nachbarn neue Distanzvektoren empfängt. Es ist daher sinnvoll, diese gemeinsam in die Neuberechnung des eigenen Distanzvektors einzubeziehen.

#### 5.4.2.2 Versenden eines Distanzvektors

Um die Anzahl der Meldungen, die nach einer Topologieänderung versendet werden, zu reduzieren, wird ein Distanzvektor nur dann an einen Nachbarknoten gesendet, wenn er sich gegenüber der letzten Version unterscheidet. Dabei ist zu beachten, daß für den Vergleich nicht nur die Länge des Wegs, sondern auch die Weginformation herangezogen werden müssen. Dazu kann folgende Situation betrachtet werden: Bei dem bisher beschriebenen Algorithmus ist die Reihenfolge, in der die Wege gleicher Länge sortiert werden, beliebig. Im Laufe der Berechnung können zwei Wege gleicher Länge in der Verkehrslenkungstabelle vertauscht werden, bzw. der erste von beiden kann ungültig werden. Obwohl sich die Entfernung des Knotens zum Ziel nicht geändert hat, muß die neue Weginformation den Nachbarn mitgeteilt werden, da deren Schleifenprüfung darauf aufbaut.

### 5.4.3 Berechnung mehrerer Alternativen

Das hauptsächliche Problem bei der Berechnung der Alternativen besteht in der Erkennung und Vermeidung von Schleifen. Um Schleifen zu erkennen, muß ein Knoten wissen, wie der weitere Weg von einem Nachbarn bis zum Zielknoten aussieht. Ist er selbst bereits Teil dieses Wegs, führt der Nachbar zu keiner gültigen Alternative.

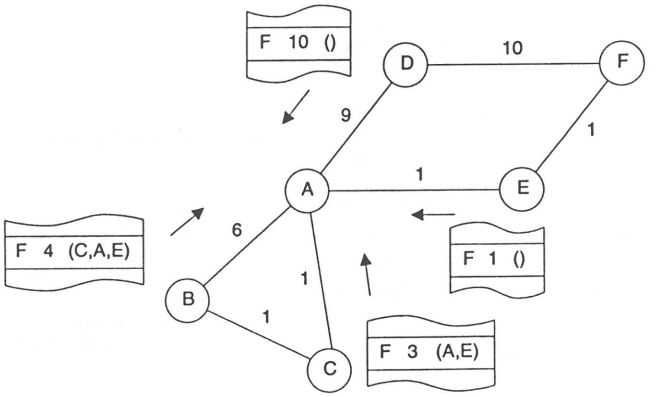



Bild 5.9: Beispiel für die Berechnung von Alternativen mit veränderten Distanzvektoren

In Abbildung 5.9 ist das Beispiel aus Abbildung 5.2 mit den neuen Distanzvektoren nochmals dargestellt. Man erkennt, daß zusätzlich zur Entfernung zu einem Zielknoten auch die Nummern der Transferknoten des entsprechenden Wegs übertragen werden. Knoten A stellt bei der Bestimmung seiner Wege fest, daß er in den Wegen, die er von den Nachbarn B und C mitgeteilt bekommt, bereits enthalten ist und berücksichtigt diese im folgenden nicht weiter. In den Wegen, die er von seinen Nachbarn D bzw. E erhält, ist A

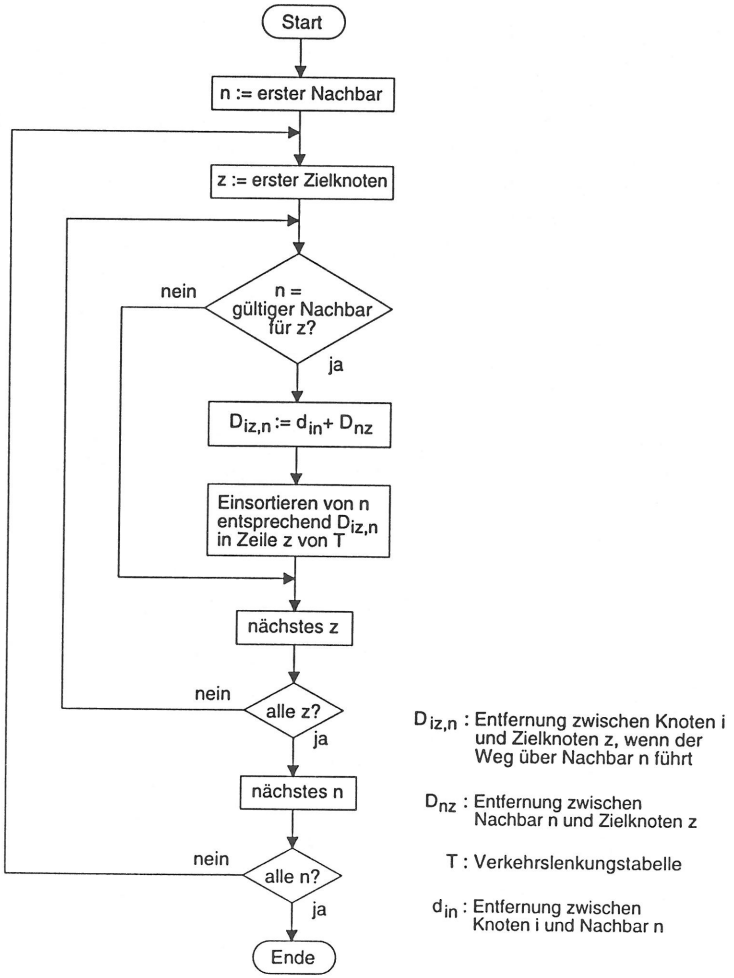



Bild 5.10: Flußdiagramm für die Berechnung der Wege in Knoten i

nicht enthalten. D und E führen folglich zu gültigen Wegen, deren Länge sich mit Hilfe von

$$D_{ik,j} = d_{ij} + D_{jk} \tag{5.1}$$

zu $D_{AF,D} = 9 + 10 = 19$ bzw. $D_{AF,E} = 1 + 1 = 2$ berechnen. Nach dem Sortieren der Längen erhält man den Erstweg über den Nachbarn E und einen Alternativweg über Knoten D. Der vollständige Algorithmus zur Berechnung aller Wege ist in Abbildung 5.10 dargestellt.

## 5.5 Optimierung des Protokolls

Das bisher vorgestellte Protokoll erfüllt zwar die gestellte Aufgabe und berechnet mehrere schleifenfreie Wege, es ist aber nicht optimal. Der größte Nachteil besteht darin, daß die Distanzvektoren für jeden Netzknoten eine fast vollständige Wegbeschreibung enthalten. Bei größeren Netzen kann dies dazu führen, daß die Protokollmeldungen groß werden und das Netz unnötig belasten.

Bei der Berechnung der Verkehrslenkungstabelle werden die Wege nach ihrer Länge sortiert. Im Vorangegangenen wurde bislang kein spezielles Sortierkriterium für den Fall angegeben, daß zwei Wege die gleiche Länge besitzen. Die Reihenfolge, in der sie letztendlich in der Verkehrslenkungstabelle stehen, ist mehr oder weniger zufällig.

Die nachfolgenden Abschnitte setzen an diesen Stellen an und optimieren den vorgestellten Algorithmus.

### 5.5.1 Optimierung der Verkehrslenkungstabellen

Wie bereits mehrfach erwähnt wurde, ist das Hauptproblem bei der Benutzung von mehreren Alternativen die Entstehung von Schleifen. Der vorgestellte Algorithmus verhindert diese Schleifenbildung. Würde die Benutzung eines Wegs zu einer Schleife führen, wird er nicht in die Verkehrslenkungstabelle aufgenommen. Wege gleicher Länge erscheinen bisher in beliebiger Reihenfolge in der Verkehrslenkungstabelle. Dies kann in manchen Fällen dazu führen, daß andere Knoten des Netzes Wege nicht in ihre Tabellen aufnehmen können, weil sonst Schleifen entstehen würden. Damit alle Netzknoten möglichst viele Alternativen in ihre Tabellen eintragen können, müssen die Tabellen aufeinander abgstimmt und eventuell umsortiert werden. Das Problem wird anhand des Beipiels in Abbildung 5.11 deutlich.

In der Abbildung sind ein einfaches Netz und Ausschnitte aus den Verkehrslenkungstabellen der Knoten A und D dargestellt. Die Tabellen wurden mit dem Algorithmus

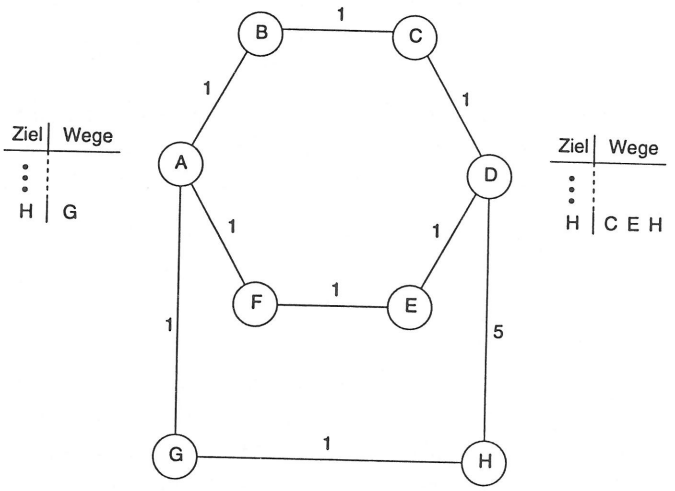



Bild 5.11: Beispielnetz für Optimierung der Verkehrslenkungstabellen

berechnet, wie er bislang vorgestellt wurde. Interessant sind die Zeilen für den Zielknoten H. Wie man leicht aus der Abbildung erkennt, gibt es in Knoten D drei Möglichkeiten, zu H zu gelangen. Dies sind die Wege (D,C,B,A,G,H), (D,E,F,A,G,H) und (D,H). Alle drei haben die gleiche Länge $D_{AH} = 5$. Im vorliegenden Fall wurden sie entsprechend ihrer Berechnungsreihenfolge in die Tabelle eingetragen (Wege über Nachbar C werden vor denen über Nachbar E berechnet). Dies führt dazu, daß Knoten A nur einen gültigen Weg nach H finden kann. Die Wege über die beiden Nachbarn B und F würden zwangsläufig zu einer Schleife führen. Da A dies erkennt, trägt er sie nicht in seine Tabelle ein.

Um zu verstehen, warum die beiden Wege zu einer Schleife führen, stelle man folgende Betrachtung an: Knoten A sendet den Verbindungswunsch weiter an Knoten B. Das Verbindungsaufbaupaket gelangt schließlich über C zu D. Knoten D überprüft nun der Reihe nach seine Einträge in der Verkehrslenkungstabelle. Der Erstweg führt direkt zurück zu C und wird deshalb von D verworfen. Er benutzt folglich den Zweitweg über E. Dieser führt aber zurück zu A. Bei der Auswahl der Wege kann D nicht unterscheiden, ob der Verbindungswunsch seinen Ursprung in A, B oder C hat. Nur für den Fall, daß A der Ursprungsknoten ist, entsteht eine Schleife. Für die beiden letzten Fälle ist die gewählte Alternative über E ein gültiger Weg.

Entsprechendes gilt für den Fall, daß A den Verbindungswunsch an seinen Nachbarn F weiterleitet. Knoten D wählt seinen Erstweg aus, da er Verbindungen mit Ursprung A nicht von denen mit Ursprung E oder F unterscheiden kann. Der gewählte Erstweg führt über C und B zurück zu A.

Das Problem ließe sich sehr einfach dadurch beheben, daß D seine gleich langen Wege in einer anderen Reihenfolge in die Verkehrslenkungstabelle einträgt. Wählt man den Direktweg (D,H) als Erstweg, kann A alle drei Alternativen benutzen.

In einer zweiter Version des Algorithmus wurde nach der Berechnung der Verkehrslenkungstabelle eine spezielle Optimierungsfunktion eingeführt. Diese sortiert Wege gleicher Länge so um, daß möglichst wenig Wege in den anderen Knoten blockiert werden. Im Gegensatz zu Link-State-Algorithmen hat ein Knoten bei einem Vector-Distance-Algorithmus keine exakte Kenntnis über die Topologie des Netzes. Er kann deshalb nicht so einfach wie in [153] die Tabellen der anderen Knoten berechnen, um so festzustellen, inwieweit diese durch die eigene Tabelle beeinflußt werden. Außerdem wäre ein solcher Ansatz äußerst rechenzeitaufwendig und sollte schon aus diesen Gründen vermieden werden.

Für die Optimierung der Tabellen werden drei zusätzliche Bewertungskriterien für Wege eingeführt. Diese sind:

- die Anzahl der Knoten, die ein Weg mit anderen gemeinsam hat,
- die Anzahl der Transferknoten,
- und die Position des Wegs in der zuletzt berechneten Tabelle.

Wege gleicher Länge werden der Reihe nach mit obigen Kriterien bewertet. Als erstes wird versucht, Wege zu bevorzugen, die sich von den kürzeren Wegen mit gleichem Ziel möglichst stark unterscheiden. Knoten, die in mehreren Wegen enthalten sind, sind „kritische“ Knoten. Bei ihnen besteht potentiell die Gefahr, daß ihre Tabellen weniger Einträge besitzen. Das Streichen von Einträgen in der Tabelle von Knoten A aus Abbildung 5.11 rührt daher, daß sowohl der Erstweg von D zu H als auch sein Zweitweg A beinhalten.

Durch das Sortierkriterium versucht der Algorithmus möglichst verschiedene Wege auszuwählen. Dies hat auch aus Sicht der Zuverlässigkeit Vorteile. Bei einem Leitungsausfall sind so weniger Wege betroffen.

Kann anhand dieses zweiten Kriteriums immer noch kein Weg bevorzugt werden, wird als nächstes Kriterium die Anzahl der Transferknoten eines Wegs herangezogen. Wege mit weniger Transferknoten werden besser bewertet.

Als letztes Kriterium wird die Position des Wegs in der zuletzt berechneten Verkehrslenkungstabelle verwendet. Der Algorithmus versucht, die Tabellen so stabil wie möglich zu halten. Die Positionen von Einträgen werden nur dann vertauscht, wenn dies einen wirklichen Vorteil bringt. Durch diese Maßnahme kann die Zahl der Meldungen reduziert werden, die zwischen den Knoten ausgetauscht werden müssen.

Die vorgestellten Sortierkriterien liefern für die meisten Netztopologien wesentlich bessere Ergebnisse, als der Algorithmus ohne Optimierung. Trotzdem gibt es vereinzelt Fälle,

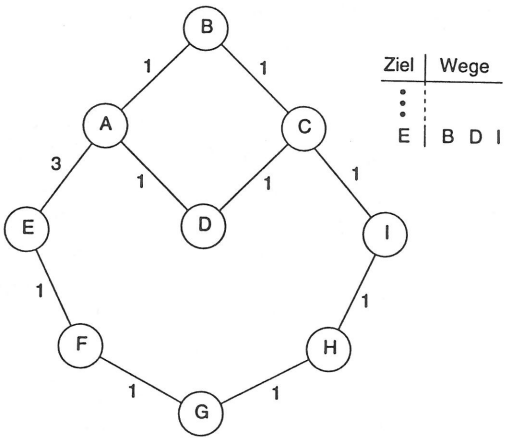



Bild 5.12: Gegenbeispiel für die Sortierkriterien

in denen die Kriterien nicht die richtige Reihenfolge der Wege finden. Eine solche Topologie ist in Abbildung 5.12 dargestellt. Knoten C hat drei mögliche Wege, um Verbindungen zu E aufzubauen. Die entsprechende Zeile aus der Verkehrslenkungstabelle ist in der Abbildung ebenfalls dargestellt. Sortiert man die Wege entsprechend der oben angegebenen Kriterien, wird der Weg über Nachbar I immer schlechter sein als die beiden anderen, da er mehr Transferknoten enthält. Die führt dazu, daß Knoten A nur einen Eintrag in seine Tabelle aufnehmen kann. Auf dieses Problem wird im nächsten Abschnitt nochmals eingegangen.

### 5.5.2 Reduzierung der Meldungsgröße

Der größte Nachteil des bisher vorgestellten Protokolls liegt in der Größe der Distanzvektoren, die im wesentlichen durch die zusätzliche Weginformation verursacht wird. Vor allem bei großen Netzen sollte die dadurch verursachte Belastung des Netzes nicht einfach vernachlässigt werden. In diesem Abschnitt soll deshalb untersucht werden, inwieweit die Größe der Distanzvektoren reduziert werden kann.

Die Weginformation wurde zu den Einträgen der Distanzvektoren hinzugefügt, um die Bildung von Schleifen zu vermeiden. Selbst wenn die Weginformation immer leer wäre, würden beim Verbindungsaufbau unmittelbare Schleifen vermieden. Ein Paket wird niemals an den Knoten zurückgesendet, von dem es empfangen wurde. Die Übertragung von Weginformation dient der Erkennung von Schleifen, die aus mehreren Knoten bestehen (vgl. Abschnitt 5.4.1).

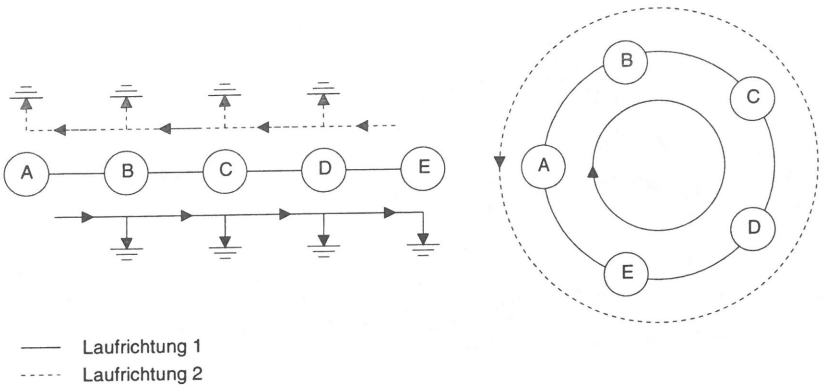



Bild 5.13: Entstehung von Schleifen an Knoten mit höchstens 2 Nachbarn

Zur Entstehung von Schleifen betrachte man Abbildung 5.13. Dort sind zwei einfache Netze mit Bus- bzw. Ringtopologie dargestellt. Jeder Knoten besitzt höchstens zwei Leitungen zu Nachbarn. Auf einer der beiden Leitungen muß er ein Paket empfangen. Da unmittelbare Schleifen beim Verbindugsaufbau erkannt und vermieden werden, darf das Paket auf dieser Leitung nicht wieder ausgesendet werden. Sofern überhaupt ein gültiger Weg zum Zielknoten existiert, muß dieser über die zweite Leitung führen. Da jeder Knoten nach diesem Prinzip handelt, gibt es bei der Busstruktur zwei Möglichkeiten: Der Zielknoten wird erreicht oder es existiert kein gültiger Weg. In letzterem Fall wird der Verbindungswunsch sofort abgelehnt (vgl. Abschnitt 5.4.1). Dies wäre z.B. der Fall, wenn Knoten C, aus welchen Gründen auch immer, ein Paket von B mit dem Ziel A erhält. Entscheidend ist, daß bei der Busstruktur keine Schleifen entstehen können. Für die Ringtopologie gilt ähnliches. Ein Paket kann hier zwar im Kreis laufen, jeder Zielknoten wird aber auf alle Fälle erreicht, bevor die Schleife geschlossen ist.

Zusammenfassend kann man festhalten, daß bei Netzformen, deren Knoten höchstens zwei Nachbarn besitzen, keine Schleifen entstehen. Diese können nur dann auftreten, wenn mindestens ein Knoten, wie in Abbildung 5.14, drei oder mehr Nachbarn besitzt. Ein Paket kommt hier auf der ersten Leitung an und wird über die zweite weitergesendet. Die Schleife entsteht, wenn es den Knoten über die dritte Leitung zum zweiten Mal erreicht. Aus diesem Grund müssen in der Weginformation der Distanzvektoren nur Knoten enthalten sein, die drei oder mehr Nachbarn besitzen.

Bei Netzen mit hohem Vermaschungsgrad hilft diese Reduktion der Weginformation nicht viel, da dort die meisten Knoten drei oder mehr Nachbarn besitzen. Allerdings kann man in diesem Fall auch davon ausgehen, daß bei einem hohen Vermaschungsgrad die Wege insgesamt sehr kurz sein werden.

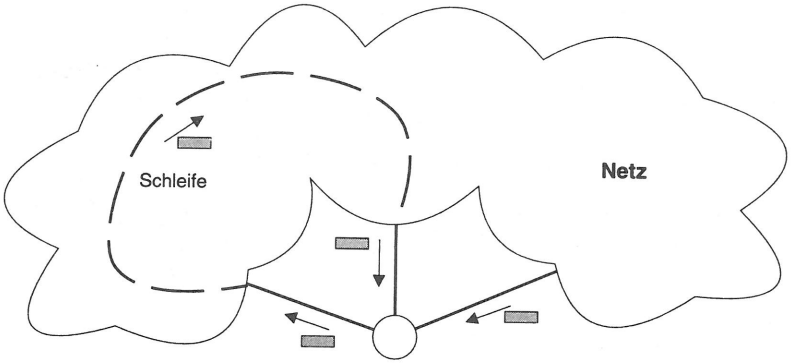



Bild 5.14: Entstehung von Schleifen an Knoten mit mindestens drei Nachbarn

Kombiniert man diese neue Berechnung der Weginformation mit der Optimierung der Verkehrslenkungstabelle des vorigen Abschnitts, ergibt sich ein weiterer positiver Effekt. Die ersten beiden zusätzlichen Bewertungskriterien (Anzahl der gemeinsamen Knoten, Anzahl der Knoten) werden anhand der Weginformation ermittelt. Hier werden nun nur noch Knoten mit drei oder mehr Nachbarn gezählt. Dies sind auch im Sinne der Optimierung die eigentlich „kritischen" Knoten. Für das Gegenbeispiel aus Abbildung 5.12 werden die Wege jetzt in der optimalen Reihenfolge in die Tabellen eingetragen.

Der verwendete Algorithmus zur Berechnung der Weginformation wurde bereits in Abbildung 5.10 vorgestellt. Wie man erkennt, trägt ein Knoten sich nicht selbst in die Weginformation ein. Er wird gegebenenfalls von seinen Nachbarn eingetragen, falls diese ihn als Transferknoten benutzen. Ein Nachbar weiß aber in der Regel nicht, wieviele Leitungen der Knoten besitzt. Der Nachbar kann deshalb nicht entscheiden, ob der vorangegangene Knoten in die Weginformation aufgenommen werden muß ($\geq 3$ Leitungen). Damit dies möglich wird, muß jeder Knoten seinen Nachbarn über einen gesonderten Eintrag im Distanzvektor mitteilen, ob er gegebenenfalls in die Weginformation aufgenommen werden soll oder nicht.

Bei genauerer Betrachtung des Algorithmus zur Wegeberechnung stellt man fest, daß zwei Zeilen des Distanzvektors keinen Einfluß auf das Ergebnis haben. Dies sind die Zeilen, die dem sendenden bzw. dem empfangenden Knoten als Ziel entsprechen. Die Entfernung des Senders zu sich selbst ist immer 0 und die zugehörige Weginformation ist leer. Der Eintrag mit der Entfernung des Senders zum Empfänger ist uninteressant, da der Empfänger nie einen Weg zu sich selbst berechnen muß.

Im Prinzip müssen diese Zeilen nicht im Distanzvektor enthalten sein. Falls sie trotzdem übertragen werden (weil dann z.B. einfacher vom Zeilenindex auf den Zielknoten

geschlossen werden kann), bietet es sich an, einen der Distanzeinträge mit dem oben erwähnten Kennzeichnungsfeld zu belegen. Ein Knoten schreibt z.B. die Anzahl seiner Leitungen in das Feld für die Entfernung zu sich selbst. Bei der Berechnung der Weginformation benutzt ein Nachbar diesen Wert für die Entscheidung, ob der Knoten aufgenommen werden muß oder nicht. Der zweite, unbenutzte Distanzeintrag sollte auf einen definierten Wert (z.B. 0 oder $\infty$) gesetzt werden. Die beiden zugehörigen Einträge für die Weginformationen sind leer.

Der neue Algorithmus zur Berechnung der Distanzvektoren ist in Abbildung 5.15 dargestellt. Auf eine erneute Darstellung des Algorithmus zur Berechnung der Verkehrslenkungstabellen wird an dieser Stelle verzichtet. Das beschriebene Umsortieren der Wege gleicher Länge wird einfach im Anschluß an die Berechnung entsprechend Abbildung 5.10 durchgeführt.

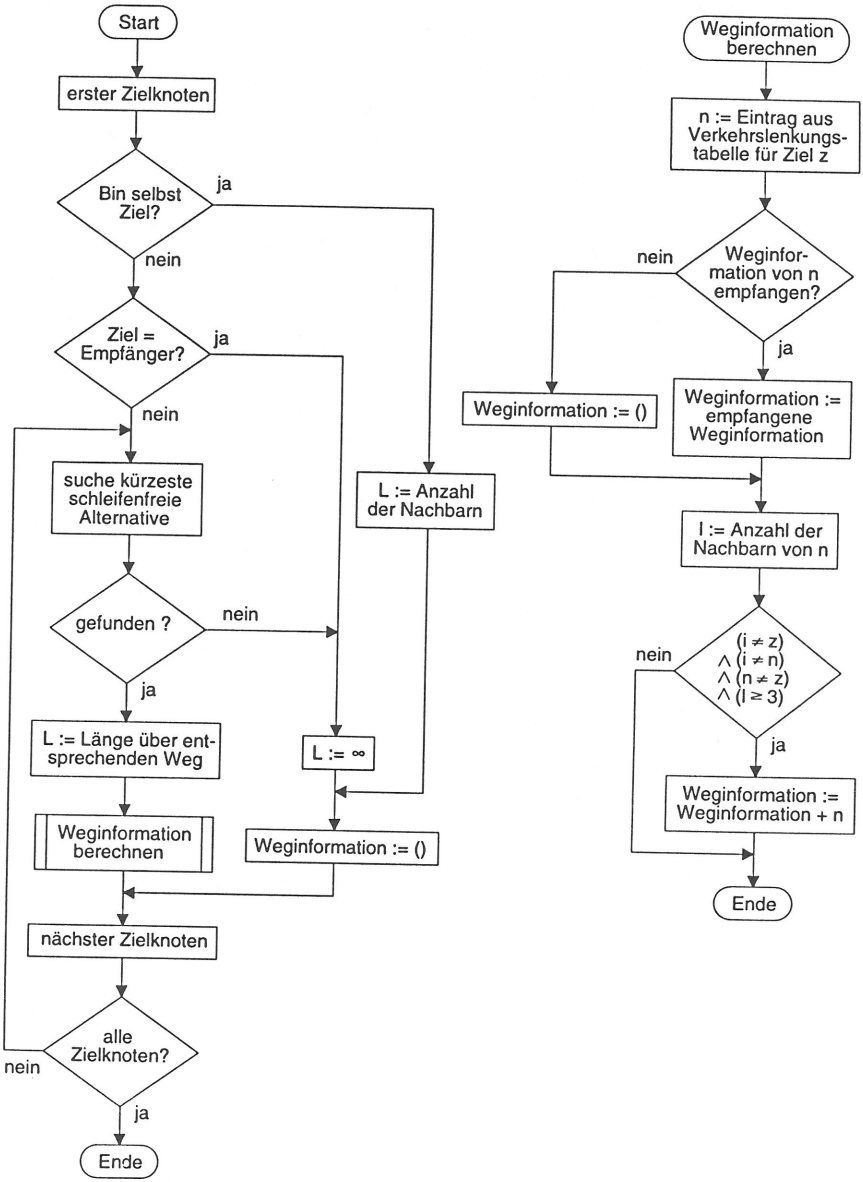



Bild 5.15: Endgültige Version des Algorithmus zur Berechnung der Distanzvektoren in Knoten i

# Kapitel 6

# Vergleich verschiedener Verfahren zur Verkehrslenkung

## 6.1 Konfiguration und Effizienzkriterien

In diesem Kapitel sollen die Eigenschaften der verschiedenen Protokollvarianten, die in Kapitel 5 entworfen wurden, genauer untersucht und mit bekannten Verfahren verglichen werden. Die Untersuchung wird mit Hilfe des Simulationswerkzeugs aus Kapitel 4 durchgeführt. Für die Bewertung und den Vergleich der Verfahren wurden folgende Kriterien herangezogen:

- Bildung von Schleifen nach einer Topologieänderung des Netzes,
- Anzahl der Knoten, die an einem Meldungsaustausch nach einer Topologieänderung beteiligt sind,
- Dauer eines Meldungsaustauschs,
- Anzahl der Knoten, die nach einer Topologieänderung zeitweilig nicht mehr erreichbar sind,
- Dauer, die einzelne Knoten nach einer Änderung nicht mehr erreichbar sind,
- Overhead, der durch den Meldungsaustausch verursacht wird.

Um einen Überblick über das Verhalten der Verfahren in verschiedenen Netzen zu erhalten, wurden sie in mehreren Netzen eingesetzt. Die entsprechenden Topologien werden in der Literatur [38, 55, 56, 57] häufig für den Vergleich von Verkehrslenkungsalgorithmen verwendet. Sie sind in Abbildung 6.1 dargestellt. Es wurde der Einfachheit halber angenommen, daß alle Leitungen die gleiche Länge besitzen.

Für die Bewertung der Verfahren wurden bei jedem Netz der Reihe nach für alle Leitungen zuerst ein Ausfall und anschließend die Inbetriebnahme simuliert. Um einen

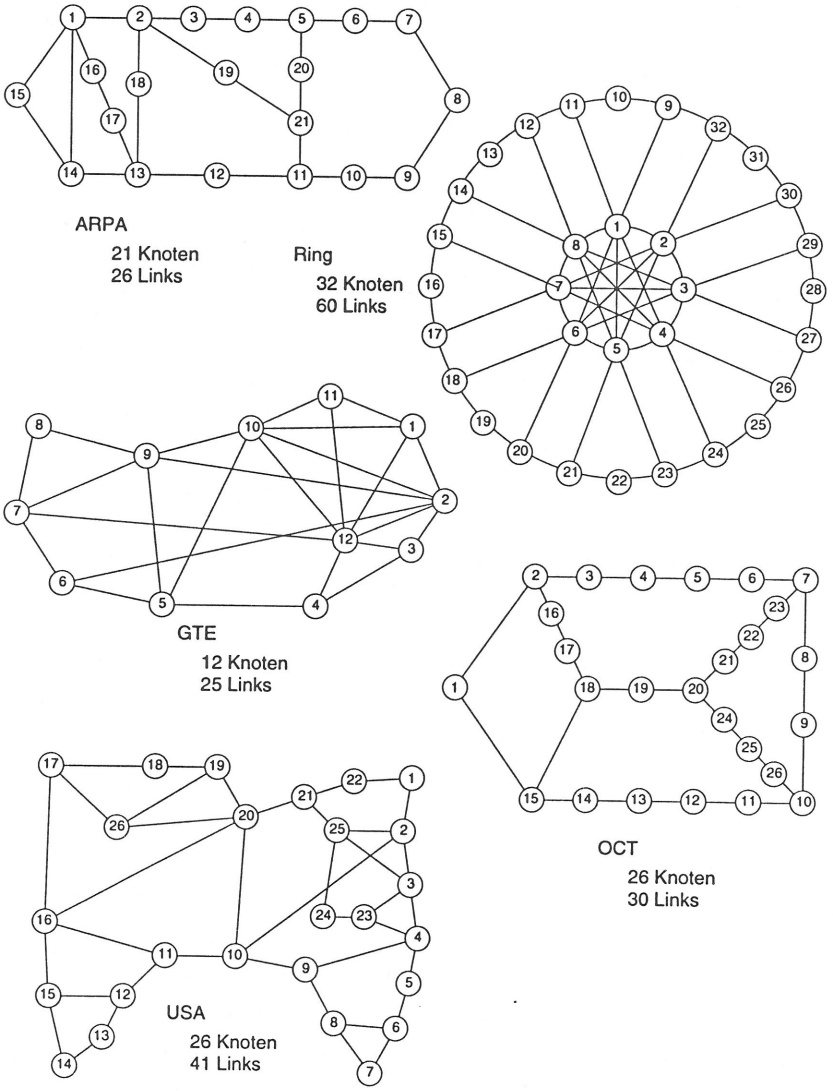



Bild 6.1: Netztopologien für die Untersuchung der Verfahren

repräsentativen Wert für die jeweiligen Meßgrößen zu erhalten, wurde über die Einzelergebnisse gemittelt. Die Simulation der Ausfälle bzw. Inbetriebnahmen wurden so durchgeführt, daß sie sich gegenseitig nicht beeinflussen.

Es wurden jeweils folgende Verfahren verglichen:

- Ein einfacher verteilter Bellman-Ford-Algorithmus, der nur den kürzesten Weg zwischen zwei Knoten und keine Alternativen bestimmt. Der Algorithmus wird in den Diagrammen mit BF abgekürzt.
- Ein Link-State-Algorithmus, der die Alternativen nach [147] berechnet. Der Algorithmus wird mit LS abgeküzt.
- Der verteilte Algorithmus, der in Abschnitt 5.4 entwickelt wurde und die vollständige Weginformationen austauscht. Er wird mit DA (distributed with alternatives) abgekürzt.
- Die Variante des Algorithmus, die eine Optimierung der Verkehrslenkungstabellen entsprechend Abschnitt 5.5.1 vornimmt. Sie wird mit OPT (optimized) abgekürzt.
- Die Variante des Algorithmus, die eine Optimierung der Meldungsgrößen entsprechend Abschnitt 5.5.2 vornimmt. Sie wird mit PR (path reduced) abgekürzt.
- Die endgültige Variante des neuen Algorithmus, die sowohl die Optimierung der Tabellen als auch die Optimierung der Meldungsgrößen vornimmt. Sie wird mit OPR (optimized and path reduced) abgekürzt.

## 6.2 Bewertung

### 6.2.1 Schleifenfreiheit

Ein sehr wesentliches Kriterium für die Güte der Algorithmen ist die Schleifenfreiheit. Zur Untersuchung, ob sich während des Ablaufs der Algorithmen Schleifen bilden können, wird nach jedem Protokollschritt versucht, für alle Kombinationen aus Quell- und Zielknoten eine Verbindung aufzubauen. Der Verbindungsaufbau wird überwacht, eventuell entstehende Schleifen werden aufgezeichnet.

Keines der Verfahren ist zu jedem Zeitpunkt grundsätzlich schleifenfrei. Die Schleifen können immer dann entstehen, wenn eine Leitung ausfällt oder sich deren Länge vergrößert. Die Schleifen sind nur temporär und lösen sich wieder auf, wenn die Algorithmen zu ihrer endgültigen Lösung konvergieren. Die Ursache für diese Schleifen liegt darin, daß sich die Information über die neue Topologie des Netzes bzw. die neue Distanzinformation schrittweise innerhalb des Netzes verbreitet. Kurz nach der Topologieänderung beruhen die Wegberechnungen eines Teils der Knoten auf veralteten Daten.

Die Schleifen, die sich beim Bellman-Ford-Algorithmus ergeben, erstrecken sich in der Regel über mehr als zwei Knoten. Beim Link-State-Algorithmus treten bei den vorgenommenen Untersuchungen nur Schleifen auf, die aus zwei Knoten bestehen, d.h. einzelne

Knoten versuchen, ein Paket zu dem Knoten zurückzuschicken, von dem sie es empfangen haben. Der Grund dafür ist in der Verbindungsaufbaustrategie zu suchen. Es wird immer nur versucht, eine Verbindung über den Erstweg aufzubauen. Ist dieser Erstweg nicht benutzbar, weil er z.B. direkt zu einer Schleife führt, wird die Verbindung sofort abgelehnt. Auf die Alternativen wird nur zurückgegriffen, wenn der Erstweg ausfällt. Will man auf die Alternativen auch ausweichen, wenn der Erstweg zwar intakt ist, aber zu einer Schleife führen würde, entstehen auch hier Schleifen mit mehr als zwei Knoten. Sollen diese vermieden werden, ist ein zusätzlicher Informationsaustausch, ähnlich wie bei dem neu entwickelten Protokoll, zwischen den Knoten notwendig.

Bei den neu entwickelten Protokollen entstehen in bestimmten Einzelfällen ebenfalls kurzzeitig Schleifen. Die Anzahl der Schleifen, die während der gesamten Untersuchungen entstehen, ist aber gegenüber den Vergleichsalgorithmen deutlich geringer. Eine genauere quantitative Auswertung erfolgt im Kapitel 6.2.3. Grundsätzlich sind im stationären Zustand alle berechneten Wege (Erstweg und Alternativen) schleifenfrei.

Betrachtet man die entstehenden Schleifen etwas genauer, stellt man folgendes fest: Bei den neu entwickelten Protokollen entstehen keine Schleifen in den Wegen, die von den Knoten ausgehen, die unmittelbar von einem Leitungsausfall betroffenen sind (d.h. die beiden Knoten, zwischen denen die Leitung ausfällt). Diese beiden Knoten können sofort den nächsten Alternativweg benutzen, um einen Zielknoten zu erreichen. Das bedeutet, daß für diese Knoten das Entwicklungsziel erreicht wird, nach einem Ausfall schnell schleifenfreie Alternativen bereitzustellen. Die Schleifen entstehen meistens in Verbindungen von Knoten, die vom Ausfallort weiter entfernt sind. Fällt z.B. beim ARPA-Netz nach Abbildung 6.1 die Leitung zwischen Knoten 2 und 3 aus, erreichen diese weiterhin alle restlichen Knoten über ihre Alternativwege. Eine vorübergehende Schleife entsteht hier aber in dem Weg von Knoten 13 zu Knoten 3. Beim Bellman-Ford und Link-State-Algorithmus sind alle Knoten mehr oder weniger gleichmäßig von der Schleifenbildung betroffen.

Sollen Schleifen erkannt werden, müssen die Pakete (Verbindungsaufbaupakete bei verbindungsorientierter Kommunikation) eine Art Spur mit sich führen. Dort werden die Knoten festgehalten, die das Paket bereits durchlaufen hat. Um den dadurch entstehenden Overhead zu reduzieren, kann an dieser Stelle die gleiche Reduktion stattfinden, wie bei der Übertragung der Weginformation in Kapitel 5.5.2.

### 6.2.2 Meldungsaustausch

In diesem Kapitel soll der Meldungsaustausch der einzelnen Protokolle nach dem Ausfall einer Leitung bzw. nach einer Längenänderung untersucht und verglichen werden.

Wichtige Kenngrößen hierfür sind die Lokalität, d.h. wieviele Knoten von dem Meldungsaustausch betroffen sind, und die Dauer des Meldungsaustauschs, die ein Maß für die Konvergenz der Algorithmen ist. Da die Einzelergebnisse von der Leitung abhängen können, die ausfällt bzw. ihre Länge ändert, wurden die Ergebnisse über alle Leitungen des Netzes gemittelt.

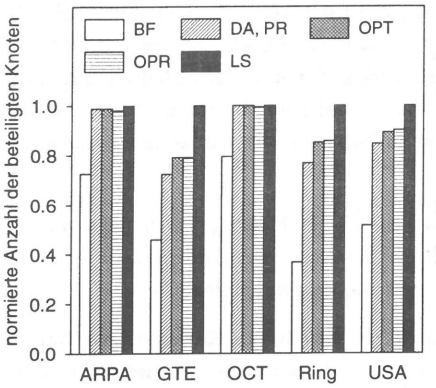



Bild 6.2: Lokalität des Meldungsaustauschs

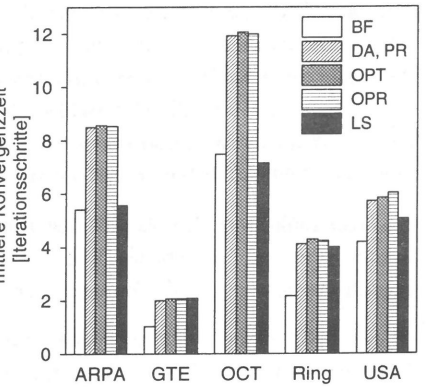



Bild 6.3: Mittlere Konvergenzzeit bei verschiedenen Netzen

Abbildung 6.2 zeigt die mittlere Anzahl der Knoten, die an einem Meldungsaustausch nach einem Ausfall und der anschließenden Wiederinbetriebnahme beteiligt waren. Die Anzahl wurde dabei mit der Gesamtzahl der Knoten des jeweiligen Netzes normiert. Beim Bellman-Ford-Algorithmus sind im Mittel am wenigsten Knoten von einem Meldungsaustausch betroffen. Die Algorithmen DA und PR verhalten sich bezüglich der Lokalität identisch. Da bei diesen Algorithmen mehrere Wege berechnet werden, sind folglich auch mehr Knoten an dem Meldungsaustausch beteiligt als beim Bellman-Ford-Algorithmus.

Die Algorithmen OPT und OPR weisen in der Regel sehr ähnliche Ergbnisse auf wie die Algorithmen DA und PR. Vereinzelt treten geringe Unterschiede sowohl zwischen OPT und OPR als auch zwischen diesen und DA bzw. PR auf. Diese sind auf die Optimierung der Tabellen zurückzuführen. Durch das Umsortieren der Alternativen werden mehr Wege verfügbar. Diese führen unter Umständen über neue Knoten, die folglich auch vom Meldungsaustausch betroffen sind.

Wie man aus der Abbildung erkennt, ist die Anzahl der beteiligten Knoten bei Link-State-Protokollen immer am größten. Da die Informationen per Broadcast versendet werden, sind hier immer alle Netzknoten am Austausch beteiligt. Die Unterschiede gegenüber den anderen Algorithmen werden um so deutlicher, je stärker die Netze vermascht sind

(z.B. GTE und Ring). Vergleichbare Resultate erhält man für den Meldungsaustausch nach einer Längenänderung.

In Abbildung 6.3 ist die mittlere Konvergenzzeit der Algorithmen für die verschiedenen Testnetze dargestellt. Die Konvergenzzeit wurde von Beginn des Meldungsaustauschs bis zu dem Zeitpunkt, an dem sich keine Protokollmeldungen mehr im Netz befinden, gemessen. Wie man sieht, werden die Meldungen bei den vier Varianten aus Kapitel 5.4 und 5.5 am längsten ausgetauscht. Entsprechend den Ergebnissen bei der Lokalität sind die Unterschiede zwischen den Varianten sehr gering. DA und PR verhalten sich erwartungsgemäß identisch; die Unterschiede bei den optimierten Varianten rühren wie oben daher, daß durch das Umsortieren mehr Alternativen bereitgestellt werden können, dies aber den Meldungsaustausch verlängert.

Beim Link-State-Protokoll ist die Konvergenzzeit unabhängig von der Leitung, die ausfällt oder ihre Länge ändert, immer konstant. Sie hängt vom Durchmesser des Netzes ab, d.h. der maximalen Entfernung zwischen zwei beliebigen Knoten.

Der Bellman-Ford-Algorithmus weist in der Regel die kürzeste Konvergenzzeit auf. Die Unterschiede zum Link-State-Algorithmus werden auch hier um so deutlicher, je stärker ein Netz vermascht ist. Beim Vergleich des Bellman-Ford-Algorithmus mit den vier neuen Varianten ist zu beachten, daß ersterer nur einen Weg berechnet. Der erhöhte Meldungsaustausch bei den neuen Algorithmen ist nötig, um die Alternativwege zu berechnen. Betrachtet man nur den Erstweg, konvergieren diese genauso schnell wie der Bellman-Ford-Algorithmus, also in der Regel schneller als die Link-State-Protokolle. Dies ist deshalb von Bedeutung, weil der weitaus größte Teil der Verbindungen über die Erstwege aufgebaut wird und nach dieser Zeitspanne stabil und schleifenfrei ist. Allerdings ist die Dauer des Meldungsaustauschs bei Link-State-Protokollen unabhängig von der Zahl der Alternativen, die berechnet werden.

### 6.2.3 Erreichbarkeit

Ein wesentlicher Gesichtspunkt bei der Entwicklung der neuen Algorithmen war eine hohe Verfügbarkeit des Netzes. Nach einem Ausfall sollten möglichst schnell Alternativen bereitstehen, über die Verbindungen geführt werden können.

In Abbildung 6.4 ist für jedes Netz die Anzahl der Leitungsausfälle aufgetragen, bei denen mindestens ein Knoten mindestens einen anderen Knoten kurzzeitig nicht mehr erreichen konnte. Die Gesamtzahl der Leitungen und damit der Maximalwert für ein Netz ist jeweils in Klammern angegeben. Die Gründe dafür, daß ein Knoten nicht mehr erreichbar ist, liegen entweder darin, daß der Verbindungswunsch abgelehnt oder eine Schleife erkannt wird. Die Algorithmen DA und PR bzw. OPT und OPR verhalten sich

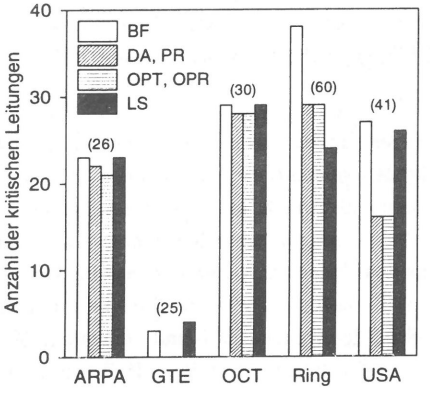



Bild 6.4: Anzahl der kritischen Leitungen bei verschiedenen Netzen

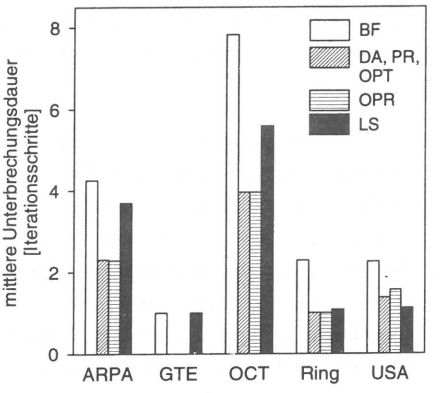



Bild 6.5: Mittlere Unterbrechungsdauer bei verschiedenen Netzen

bezüglich dieser Größe jeweils gleich. Dies war zu erwarten, da das Entfernen überflüssiger Weginformation keinen Einfluß auf die Berechnung der Wege haben sollte. Wie man sieht, sind die Ergebnisse für die Algorithmen DA und OPR in den meisten Fällen ebenfalls identisch. Nur beim ARPA-Netz ergibt sich eine leichte Verbesserung von OPR gegenüber DA. Durch die Optimierung der Tabellen können mehr Alternativen bereitgestellt werden. Dadurch müssen in bestimmten Situationen Verbindungen nicht abgelehnt werden.

Der Bellman-Ford-Algorithmus weist bei allen Netzen das schlechteste Verhalten auf. Dies ist nicht verwunderlich, da hier keine Alternativen berechnet werden. Man sieht daran deutlich, daß die vorgeschlagene Architektur mit mehreren Alternativwegen Vorteile gegenüber Verfahren mit nur einem Weg hat.

Die neu entwickelten Algorithmen verhalten sich in der Regel ebenfalls günstiger als der Link-State-Algorithmus, der mehrere Alternativen berechnet. Dies ist im wesentlichen auf die Reduzierung der Schleifenbildung zurückzuführen. Bei dem Link-State-Algorithmus stehen häufig zwar Alternativwege zur Verfügung, diese können aber nicht benutzt werden, da sie Schleifen beinhalten. Einzige Ausnahme bildet das sehr regelmäßig aufgebaute Ringnetz. Auf Grund der Berechnungsreihenfolge der Alternativen bevorzugt der Link-State-Algorithmus bei gleicher Länge Wege über die inneren Knoten (1–8). Fällt eine Leitung im äußeren Kreis (z.B. zwischen den Knoten 23 und 24) aus, dann hat dies nur einen geringen Einfluß auf die Verbindungsführung, da die meisten Verbindungen nach wie vor über innere Knoten gelegt werden. Der OPR-Algorithmus kennt diese Bevorzugung nicht, d.h. wenn eine Leitung im äußeren Ring ausfällt, sind davon mehr Verbindungen betroffen und können unter Umständen nicht mehr aufgebaut werden. Diese Bevorzugung der inneren Knoten beim Link-State-Algorithmus ist zufällig und rührt daher, daß Wege

über Knoten mit kleineren Knotennummern zuerst berechnet werden. Wie man aus den anderen Netztopologien sieht, liefert der OPR-Algorithmus in den meisten Fällen die besseren Resultate.

In Abbildung 6.5 ist die mittlere Dauer einer Unterbrechung des Netzes für die verschiedenen Netze aufgezeichnet. Damit ist die Zeit gemeint, die zwischen dem Zeitpunkt vergeht, an dem der erste Knoten eine Verbindung nicht mehr aufbauen kann, bis zu dem Zeitpunkt, an dem alle Knoten wieder alle Verbindungen aufbauen können. Wie man sieht, schneidet der Bellman-Ford-Algorithmus auch hier am schlechtesten ab. Die neu entwickelten Algorithmen liefern in der Regel die besten Ergebnisse, d.h. die kürzesten Unterbrechungszeiten. Die Unterschiede bei den Algorithmen OPT und OPR beim Netz USA rühren daher, daß die Alternativen in Einzelfällen in einer anderen Reihenfolge sortiert werden (es werden nur noch Knoten mit 3 oder mehr Leitungen berücksichtigt; vgl. Abschnitt 5.5.2, Seite 104).

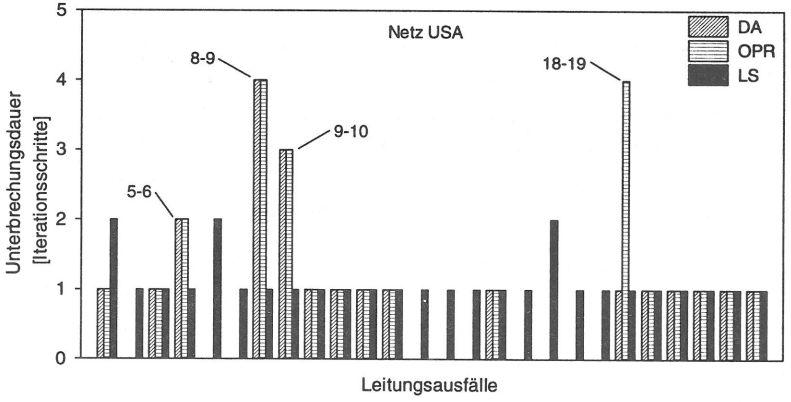



Bild 6.6: Unterbrechungsdauer bei den verschiedenen Ausfällen

In Abbildung 6.6 ist die Unterbrechungsdauer für die jeweiligen Leitungsausfälle aufgezeichnet. Wie man daraus deutlich erkennt, hat man bei den Algorithmen DA und OPR weniger Unterbrechungen als beim Link-State-Algorithmus. Der etwas höhere Mittelwert aus Abbildung 6.5 rührt daher, daß bei einzelnen Ausfällen die Unterbrechungsdauer etwas länger ist. Im Gegensatz dazu ist diese beim Link-State-Algorithmus nahezu konstant bei 1.

In Abbildung 6.7 ist die Anzahl der kritischen Leitungsausfälle aus Sicht des Knotens 1 für die verschiedenen Versuchsnetze aufgetragen. Auch in diesem Diagramm geben die geklammerten Werte die Gesamtzahl der Leitungen eines Netzes und damit den jeweiligen Maximalwert an. Man erkennt, daß die entworfenen Algorithmen bei allen Netzen deutlich

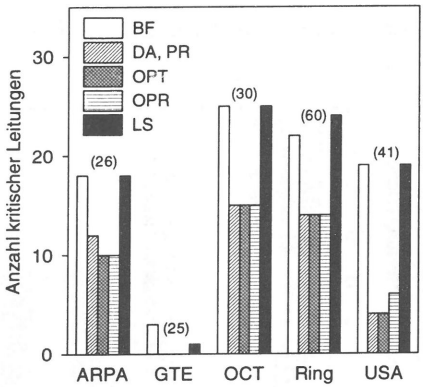



Bild 6.7: Anzahl der kritischen Leitungen aus Sicht von Knoten 1 bei verschiedenen Netzen

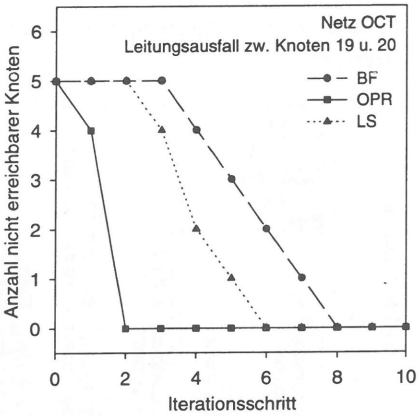



Bild 6.8: Anzahl der von Knoten 1 nicht erreichbaren Knoten

bessere Ergebnisse liefern als die Vergleichsalgorithmen Bellman-Ford und der Link-State-Algorithmus mit mehreren Alternativen. Das bedeutet, daß es weniger Leitungen gibt, bei deren Ausfall Knoten 1 Verbindungen zu anderen Netzknoten nicht aufbauen kann. Beim GTE Netz z.B. können alle Verbindungen zu jedem Zeitpunkt immer aufgebaut werden, unabhängig davon, ob und welche Leitung ausfällt. Wie bei den bisherigen Ergebnissen auch, verhalten sich die Algorithmen DA, PR, OPT und OPR sehr ähnlich. Zeitweise treten geringe Unterschiede auf Grund der optimierten Tabellen auf, die aber vergleichsweise gering gegenüber den Verbesserungen zum Bellman-Ford bzw. Link-State-Algorithmus sind.

Abbildung 6.8 zeigt beispielhaft die Anzahl der von Knoten 1 aus nicht erreichbaren Knoten im Verlauf einer Iteration für die Algorithmen BF, OPR und LS. Es wurde angenommen, daß zum Zeitpunkt 0 (Iterationsschritt 0) die Leitung zwischen den Knoten 19 und 20 des Netzes OCT ausfällt. Unmittelbar nach dem Ausfall können bei allen Algorithmen gleich viele Verbindungen nicht aufgebaut werden. Der Bellman-Ford-Algorithmus benötigt am längsten, bis Knoten 1 wieder alle anderen Netzknoten erreichen kann. Der optimierte Algorithmus mit seinen Alternativen zeigt auch hier das günstigste Verhalten.

### 6.2.4 Protokolloverhead

Ein letzter wichtiger Punkt für den Vergleich der Algorithmen ist der Overhead, der durch den Meldungsaustausch verursacht wird. In Abbildung 6.9 und 6.10 ist die mittlere Anzahl

der ausgetauschten Meldungen bzw. der mittlere Overhead pro Meldung für die Testnetze dargestellt.

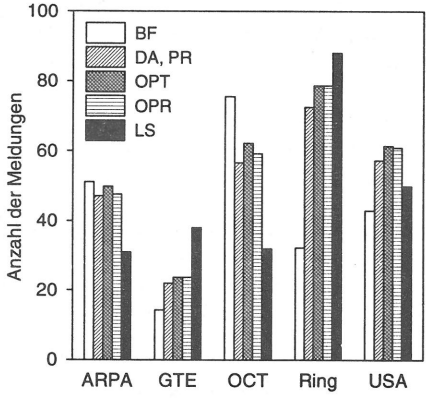



Bild 6.9: Mittlere Anzahl der ausgetauschten Meldungen

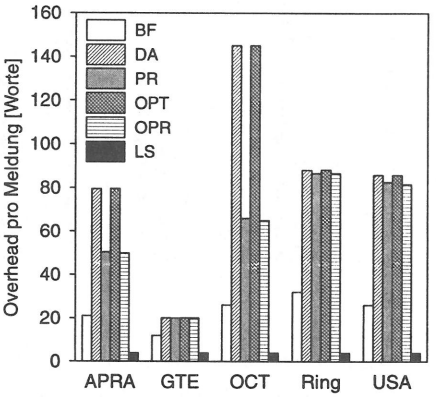



Bild 6.10: Mittlerer Overhead pro Meldung

Aus Abbildung 6.9 sieht man, daß die Anzahl der Meldungen stark von der jeweiligen Netztopologie abhängt. Die Unterschiede zwischen den vier Varianten DA, PR, OPT und OPR sind auch hier wieder minimal. Beim Link-State-Algorithmus ist die Anzahl der Meldungen konstant und wird nur durch die Zahl der Leitungen im Netz bestimmt. Bei weniger stark vermaschten Netzen werden durch das Link-State-Protokoll weniger Meldungen ausgetauscht als bei den anderen Algorithmen. Steigt der Vermaschungsgrad an, wie z.B. bei den Netzen GTE oder Ring, dann kann die Anzahl auch größer werden als bei den restlichen Algorithmen.

Der Overhead pro Meldung ist beim Link-State-Protokoll eindeutig am geringsten, da pro Fehler im Prinzip nur vier Worte (2 Knotennummern zur Identifikation einer Leitung, Länge und Folgenummer) übertragen werden müssen. Wie nicht anders zu erwarten war, steigt der Overhead pro Meldung bei den neuen Algorithmen gegenüber dem Bellman-Ford-Algorithmus stark an. Dies ist auf die zusätzliche Weginformation zurückzuführen. Man erkennt auch, daß die Optimierung der Weginformation speziell bei den Netzen ARPA und OCT den Overhead stark reduziert. Diese Netze sind nicht so stark vermascht und weisen längere busartige Teilstücke auf. Bei stärker vermaschten Netzen sind die Unterschiede dagegen nicht so groß, da hier die meisten Knoten mehr als 2 Leitungen besitzen und deshalb in der Weginformation mit übertragen werden müssen. Bei solchen Netzen sind die Wege in der Regel aber auch tendentiell kürzer, so daß die Unterschiede zum Bellman-Ford-Algorithmus, der keine Weginformation mit überträgt, nicht ganz so groß sind. Man sieht dies sehr gut beim GTE-Netz. Daß diese Aussage nicht immer gilt,

sieht man am Netz USA. Dort besitzen die meisten Knoten mehr als 2 Nachbarn, die Wege sind aber trotzdem noch relativ lang. Die Schlußfolgerung, die man daraus ziehen kann, ist, daß es durchaus sinnvoll ist, den Verkehrslenkungsalgorithmus bei der Netzplanung (Topologieplanung) zu berücksichtigen.

Beim Vergleich des Protokolloverheads ist zu beachten, daß die obigen Werte jeweils für Einzeländerungen gelten. Wird ein Meldungsaustausch nur sehr selten durchgeführt, spielt der Overhead keine so große Rolle. Ändern sich die Längen der Leitungen dagegen häufig, steigt die Anzahl der Meldungen beim Link-State-Algorithmus linear an. Bei den Distance-Vector-Algorithmen (Bellman-Ford, DA, PR, OPT, OPR) dagegen können mehrere Änderungen in einem Distanzvektor zusammengefaßt werden, so daß sich der Overhead nicht oder nur sehr wenig ändert. Ferner wurde in den dargestellten Diagrammen nicht berücksichtigt, daß bei einem Link-State-Protokoll die Topologietabellen in den Knoten aus Sicherheitsgründen in regelmäßigen Abständen ausgetauscht und miteinander verglichen werden sollten.

### 6.2.5 Adaption an Lastschwankungen

Die entwickelten Algorithmen sind grundsätzlich dazu geeignet, die Verkehrslenkung lastadaptiv durchzuführen. Der Schlüssel hierzu liegt in der Wahl einer geeigneten Metrik, die die Lastverhältnisse in Leitungslängen abbildet. Eine Laständerung resultiert dann in einer Änderung der Leitungslänge, die wiederum einen Meldungsaustausch initiiert. Die Wahl einer geeigneten Metrik ist dabei unabhängig von der Berechnung der Wege zu sehen.

Die oben beschriebenen Untersuchungen wurden auch mit Längenänderungen der Leitungen durchgeführt und führten zu prinzipiell ähnlichen Ergebnissen, da ein Ausfall einer Leitung mit einer stark überlasteten Leitung und damit einer Längenänderung auf einen großen Wert gleichzusetzen ist. Es wurde allerdings angenommen, daß die jeweiligen Längenänderungen unabhängig voneinander sind. Soll die Belastung des Netzes berücksichtigt werden, gilt diese Unabhängigkeitsannahme nicht mehr. Da sich in diesem Fall aber sehr viele Leitungslängen gleichzeitig ändern, ist zu erwarten, daß die Algorithmen in Bezug auf den Protokolloverhead deutlich günstigere Ergebnisse liefern.

# Kapitel 7

# Zusammenfassung und Ausblick

## 7.1 Zusammenfassung

In der vorliegenden Arbeit wurde ein neues Verfahren zur adaptiven Verkehrslenkung entwickelt, das es ermöglicht, sehr schnell auf Änderungen in der Topologie eines Netzes zu reagieren. Es basiert auf der Bereitstellung mehrerer Alternativwege, auf die sofort ausgewichen werden kann, wenn der Erstweg überlastet ist oder ausfällt.

Um eine optimale Arbeitsweise zu erreichen, müssen folgende Komponenten aufeinander abgestimmt werden:

- das Protokoll als Teil des Verkehrslenkungsmanagements, das die zur Bestimmung der Wege notwendigen Daten zur Verfügung stellt,
- das Verkehrslenkungsschema, das die Bestimmung der Alternativen vornimmt,
- die Wegeauswahl, die einen Weg aus mehreren zur Verfügung stehenden Alternativen auswählt, wenn eine Verbindung aufgebaut bzw. wenn bei der verbindungslosen Kommunikation ein Paket vermittelt werden soll

In der Arbeit wurden für jede dieser Komponenten verschiedene Realisierungen in Betracht gezogen und gegeneinander abgewogen. Schließlich wurden folgende grundsätzliche Entscheidungen getroffen: Die Auswahl eines Wegs aus den zur Verfügung stehenden Alternativen erfolgt abschnittsweise, und die Berechnung mehrerer Alternativwege wird verteilt durchgeführt.

Für die Bestimmung mehrerer Alternativen wurde ein neuer Shortest Path-Algorithmus entwickelt. Dieser berücksichtigt bei der Berechnung der Alternativen bereits die Strategie, mit der später ein Weg bei einem Verbindungsaufbau bzw. bei der Vermittlung eines Pakets ausgewählt wird. Dadurch konnte erreicht werden, daß alle Alternativwege

schleifenfrei sind. Fällt ein Erstweg aus, kann sofort auf eine Alternative ausgewichen werden. Im Gegensatz zu den bisher aus der Literatur bekannten Verfahren müssen Pakete, die über eine Alternative gelenkt werden, nicht besonders gekennzeichnet werden.

Um die Wege berechnen zu können, tauschen die Netzknoten untereinander Informationen aus. Für diesen Zweck wurde ein neues Protokoll entworfen, das zur Klasse der Distance-Vector-Protokolle gehört. Jeder Netzknoten sendet an alle seine unmittelbaren Nachbarn sogenannte Distanzvektoren. In diesen sind die Entfernungen des Knotens zu jedem möglichen Zielknoten und Informationen über den zugehörigen Weg eingetragen. Die Weginformationen werden von den Nachbarknoten dazu verwendet, Schleifen zu vermeiden.

Ausgehend von einer Grundversion, wurden sowohl der Algorithmus zur Bestimmung der Wege als auch das Protokoll zum Informationsaustausch zwischen den Knoten optimiert. Durch geeignetes Sortieren von Wegen gleicher Länge konnte die Anzahl der schleifenfreien Alternativwege in den Verkehrslenkungstabellen erhöht werden. Die Optimierung des Protokolls zum Austausch der Distanzvektoren ermöglichte eine Reduzierung des durch den Austausch verursachten Verwaltungsaufwands.

Abschließend wurde das entwickelte Verkehrslenkungsverfahren simulativ mit bekannten Verfahren verglichen. Die Anzahl der temporären Schleifen, die nach einer Topologieänderung des Netzes entstehen können, ist bei dem entwickelten Verfahren deutlich geringer. Die Alternativwege der Knoten, deren Erstweg ausfällt, waren zu jedem Zeitpunkt schleifenfrei. Temporäre Schleifen entstanden nur noch vereinzelt bei Wegen, bei denen sich der ausgefallene Abschnitt in der Mitte des Wegs befand.

Die Verfügbarkeit des Netzes nach einer Topologieänderung konnte durch das neue Verfahren erhöht werden. Von einem Leitungsausfall sind weniger Quelle-Ziel-Paare betroffen als bei anderen Verfahren. Falls einzelne Zielknoten nach einer Änderung nicht mehr erreicht werden können, ist die Dauer dieser Unterbrechung bei dem neuen Verfahren in der Regel kürzer. Die Eigenschaften des Verfahrens können durch eine entsprechend gewählte Netztopologie noch verbessert werden.

Der Verwaltungsaufwand, der durch den Austausch der Distanzvektoren zwischen den Knoten verursacht wird, ist gegenüber dem Verwaltungsaufwand bei den Vergleichsverfahren größer.

Zur Untersuchung der Verkehrslenkungsverfahren wurde in dieser Arbeit ein spezieller Protokollsimulator entwickelt. Der Entwurf erfolgte unter Anwendung objektorientierter Methoden. Der Simulator basiert auf einer objektorientierten Simulationsbibliothek [87]. Um den speziellen Anforderungen bei der Untersuchung verteilter Verkehrslenkungsalgorithmen gerecht zu werden, wurde die Bibliothek erweitert und um zusätzliche Module ergänzt:

- Der Anwender kann mit Hilfe einer neuen Ablaufsteuerung interaktiv simulieren. Er hat die Möglichkeit, über eine Menüoberfläche Unterbrechungspunkte (Breakpoints) zu setzen, Simulationsereignisse im Einzelschrittbetrieb abzuarbeiten oder den Systemzustand zu jedem Zeitpunkt abzufragen oder zu verändern.
- Für die einfache Eingabe einer Netztopologie wurde ein spezieller Parser entwickelt. Dieser liest die Netztopologie und die Parameter der Netzknoten und Leitungen ein und baut ein entsprechendes Simulationsmodell automatisch auf. Beim Entwurf des Parsers wurde darauf geachtet, daß er auch für andere Programme verwendet werden kann. Durch Ableiten weniger anwendungsspezifischer Knoten- und Leitungsklassen können von ihm völlig andere Simulationsmodelle aufgebaut oder Topologiedaten für ein Analyseprogramm eingelesen werden.
- Zur Überwachung des Protokollablaufs wurde ein eigenes Konzept entwickelt und implementiert. Es erlaubt dem Anwender selektiv Meldungen auszugeben, anhand derer der Ablauf der Simulation verfolgt werden kann. Die Steuerung der Ausgabe kann dynamisch zur Laufzeit erfolgen. Das Konzept wurde so gekapselt, daß es ohne Änderungen in andere Simulationsprogrammes oder auch als Debugging-Hilfe für allgemeine Softwareprojekte übernommen werden kann.

Der Einsatz objektorientierter Methoden beim Entwurf des Simulators hat sich sehr bewährt. Sie ermöglichten in jeder Beziehung eine hohe Flexibilität und eine leichte Erweiterbarkeit. Besonders das iterative Vorgehen bei der Entwicklung objektorientierter Software war von großem Nutzen. Neue Algorithmen ließen sich sehr leicht auch nachträglich hinzufügen. Dazu war nur das Ableiten jeweils einer neuen Klasse und das Überschreiben weniger Methoden erforderlich. Da bestehender Code nicht modifiziert, sondern nur an wenigen Stellen neuer Code hinzugefügt werden mußte, war auch der Aufwand für den Test des Simulators nach einer Erweiterung deutlich geringer.

## 7.2 Ausblick

Das in der Arbeit vorgestellte Verfahren berechnet die Wege auf Grund von „Längen“, die jeder einzelnen Leitung zugewiesen werden. Die Zuordnung einer „Länge“ zu einer Leitung erfolgt mit Hilfe einer beliebigen Metrik, die Leitungs-, Verkehrs- oder Netzparameter auf entsprechende Längenwerte abbildet. Das entwickelte Verfahren trifft keinerlei Annahmen über das Aussehen einer Metrik, d.h. welche Parameter in einer Leitungslänge berücksichtigt werden. In Abschnitt 3.3.2 wurden bereits einige Beispiele für Metriken aufgezählt, die in Paketvermittlungsnetzen verwendet werden können. Von besonderem Interesse ist die Frage, wie eine Metrik definiert werden muß, wenn das Verfahren in anderen Netzen eingesetzt werden soll.

Im zukünftigen Breitband-ISDN, das auf der ATM-Technik basieren soll, wird zwischen dem physikalischen Netz und der Verbindungsebene eine logische Netzebene aus sogenannten virtuellen Pfaden eingeschoben. Verbindungen werden über virtuellen Pfade gelenkt. Ein virtueller Pfad kann mehrere physikalische Leitungen und die dazwischenliegenden Knoten umfassen. Eine Aufgabe für die Zukunft könnte darin bestehen, eine geeignete Metrik zu finden, die die spezifischen Netz- und Verkehrsparameter eines ATM-Netzes in „Leitungslängen" abbildet. Das vorgestellte Verfahren könnte dann dazu verwendet werden, die virtuellen Pfade zu berechnen.

In der Arbeit wurden bereits erste Untersuchungen zu einer lastadaptiven Verkehrslenkung durchgeführt. Dabei wurden die Längen der einzelnen Leitungen unabhängig voneinander geändert. Eine spezielle Metrik, die die tatsächliche Belastung einer Leitung in eine entsprechende Länge abbildet, wurde nicht berücksichtigt. Eine genauere Modellierung der Verkehrslast und deren Einflüsse auf die Verkehrslenkung könnte ein weiteres Ziel zukünftiger Arbeiten sein.

Der entwickelte Algorithmus bestimmt für jedes Quelle-Ziel-Paar mehrere Alternativwege. Diese werden hauptsächlich dann benutzt, wenn der Erstweg ausfällt. Es ist aber auch denkbar, daß bereits auf einen Alternativweg ausgewichen wird, wenn die Belastung des Erstwegs einen Schwellwert überschreitet. Damit könnte die „Schwingneigung" lastadaptiver Shortest Path-Algorithmen gedämpft werden. Das Erarbeiten einer Strategie, wann auf eine Alternative ausgewichen wird, ist somit eine weitere interessante Aufgabenstellung.

# Literaturverzeichnis


[1] AFEK Y., AWERBUCH B., GAFNI E.: *Applying static network protocols to dynamic networks*, Proceedings of the 28th Annual Symposium on Foundations of Computer Science, Oktober 1987.

[2] AHO A. V., ULLMAN J. D.: *Foundations of Computer Science*, Computer Science Press, 1992.

[3] ALEXANDER C.: *The Timeless Way of Building*, Oxford University Press, 1979.

[4] ARNOLDI L.: *Fast Packet Switching, Grundlagen und Technologie*, Datacom, Vol. 9, Nr. 5, Mai 1992, Seite 78–82.

[5] ASH G. R., CARDWELL R. H., MURRAY R. P.: *Design and Optimization for Networks with Dynamic Routing*, Bell System Technical Journal, Vol. 60, Nr. 8, Oktober 1981, Seite 1787–1820.

[6] ASH G. R., KAFKER A. H., KRISHNAN K. R.: *Servicing and Real-Time Control of Networks with Dynamic Routing*, Bell System Technical Journal, Vol. 60, Nr. 8, Oktober 1981, Seite 1821–1845.

[7] ASH G. R.: *Design and Control of Networks with Dynamic Nonhierarchical Routing*, IEEE Communications Magazine, Vol. 28, Nr. 10, Oktober 1990, Seite 34–40.

[8] ASH G. R.: *Use of a Trunk Status Map for Real-Time DNHR*, Proceedings of ITC 11, Kyoto, Japan, 1985, Paper 4.4A–4.

[9] AWERBUCH B.: *Shortest Paths and Loop-Free Routing in Dynamic Networks*, ACM Computer Communication Review, Vol. 20, Nr. 4, September 1990, Seite 177–187.

[10] AWERBUCH B., BAR-NOY A., GOPAL M.: *Approximate Distributed Bellman-Ford Algorithms*, Proceedings of IEEE Infocom'91, Vol. 3, Bal Harbour FL, April 1991, Seite 1206–1213.

[11] BAHK S., EL ZARKI M.: *Dynamic Multi-path Routing and How it Compares with other Dynamic Routing Algoritms for High Speed Wide Area Networks*, ACM Computer Communication Review, Vol. 22, Nr. 4, Oktober 1992, Seite 53–64.

[12] BARAN P.: *On Distributed Communication Networks*, IEEE Transactions on Communication Systems, Vol. 12, Nr. 3, März 1964, Seite 1–9.

[13] BEL G., ET. AL.: *Adaptive Traffic Routing in Telephone Networks*, Large Scale Systems Journal, Vol. 8, Nr. 3, 1985.

[14] BELL P., JABBOUR K.: *Review of Point-to-Point Network Routing Algorithms*, IEEE Communications Magazine, Vol. 24, Nr. 1, Januar 1986, Seite 34–38.

[15] BELL P., JABBOUR K.: *Evaluation of Point-to-Point Network Routing Algorithms*, IEEE Transactions on Communications, Vol. 35, Nr. 4, April 1987, Seite 470–472.

[16] BERTSEKAS D. P.: *Distributed Dynamic Programming*, IEEE Transactions on Automatic Control, Vol. 27, Nr. 3, Juni 1982, Seite 610–616.

[17] BERTSEKAS D. P., GAFNI E. M., GALLAGER R. G.: *Second Derivative Algorithms for Minimum Delay Distributed Routing in Networks*, IEEE Transactions on Communications, Vol. 32, Nr. 8, August 1984, Seite 911–919.

[18] BERTSEKAS D. P., GALLAGER R.: *Routing in Data Networks*, Data Networks, Kapitel 5, Prentice Hall International, 1987, Seite 297–422.

[19] BHARATH-KUMAR K., JAFFE J. M.: *Routing to Multiple Destinations in Computer Networks*, IEEE Transactions on Communications, Vol. 31, Nr. 3, März 1983, Seite 343–351.

[20] BOOCH G.: *Object-Oriented Analysis and Design with Applications*, Second Edition, Benjamin Cummings, 1994.

[21] BOORSTYN R., LIVNE A.: *A Technique for Adaptive Routing in Networks*, IEEE Transactions on Communications, Vol. 29, Nr. 4, April 1981, Seite 474–480.

[22] BRAESS D.: *Über ein Paradoxon aus der Verkehrsplanung*, Unternehmensforschung, Vol. 12, 1968, Seite 258–268.

[23] VAN BREEMEN J.: *Options for the MAC protocol in a Broadband PON*, Proceedings of the RACE Open Workshop on Broadband Access (BAF), Nijmegen, Niederlande, Juni 1993, Seite 6.

[24] BROOKS F. P.: *No Silver Bullet: Essence and Accidents of Software Engineering*, IEEE Computer, Vol. 20, Nr. 4, April 1987, Seite 10–19.

[25] CARDELLI L., WEGNER P.: *On Understanding Types, Data Abstraction and Polymorphism*, ACM Computing Surveys, Vol. 17, Nr. 4, Dezember 1985, Seite 471–522.

[26] CARON F.: *Results of the Telecom Canada High Performance Routing*, Proceedings of ITC 12, Turin, Italien, 1988, Paper 2.2A.2.

[27] CASSANDRAS C. G., ABIDI V. M., TOWSLEY D.: *Distributed Routing with On-Line Marginal Delay Estimation*, IEEE Transactions on Communications, Vol. 38, Nr. 3, März 1990, Seite 348–359.

[28] CEGRELL T.: *A Routing Procedure for the TIDAS Message-Switching Network*, IEEE Transactions on Communications, Vol. 23, Nr. 6, Juni 1975, Seite 575–585.

[29] CHEMOUIL P., FILIPIAK J., GAUTHIER P.: *Performance Issues in the Design of Dynamically Controlled Circuit-Switched Networks*, IEEE Communications Magazine, Vol. 28, Nr. 10, Oktober 1990, Seite 90–95.

[30] CHENG C., RILEY R.; KUMAR S.: *A Loop-Free Extended Bellman-Ford Routing Protocol without Bouncing Effect*, ACM Computer Communication Review, Vol. 19, Nr. 4, September 1989, Seite 224–236.

[31] CHOU W., BRAGG A., NILSSON A.: *The Need for Adaptive Routing in the Chaotic and Unbalanced Traffic Environment*, IEEE Transactions on Communications, Vol. 29, Nr. 4, April 1981, Seite 481–490.

[32] CHU W. W., SHEN M. Y.: *A Hierarchical Routing and Flow Control Policy (HRFC) for Packet Switched Networks*, Computer Performance, Editoren: Chandy K. M. Reiser M., North Holland Publishing 1977, Seite 485–498.

[33] COHEN A., MRABET R.: *AMS: An Integrated Simulator for Open Systems*, ULB-Rapport STC 93-28, Faculé des Sciences, Université Libre de Bruxelles, Nr. 28, Oktober 1993.

[34] COHEN J. E.: *The Counterintuitive in Conflict and Cooperation*, American Scientist, Vol. 76, 1988, Seite 576–584.

[35] COPLIEN J. O.: *Advanced C++, Programming Styles and Idioms*, Addison Wesley, 1992.

[36] COPLIEN J. O.: *Generative Pattern Languages, An emerging direction of software Design*, C++ Report, Vol. 6, Nr. 6, Juli-August 1994, Seite 18ff.

[37] COPLIEN J.O.: *Setting the Stage*, C++ Report, Vol. 6, Nr. 8, Oktober 1994, Seite 8-16.

[38] COURTOIS P.-J., SEMAL P. *An Algorithm for the Optimization of Nonbifurcated Flows in Computer Communication Networks*, Performance Evaluation, Vol. 1, 1981, Seite 139—152.

[39] DEO N., PANG C.: *Shortest Path Algorithms: Taxanomy and Annotation*, Networks, Vol. 14, 1984, Seite 275–323.

[40] DIJKSTRA E. W.: *A Note on Two Problems in Connection with Graphs*, Numerische Mathematik, Vol. 1, 1959, Seite 269–271.

[41] FINN G. S.: *Resynch Procedures and a Fail-Safe Network Protocol*, IEEE Transactions on Communications, Vol. 27, Nr. 6, Juni 1979, Seite 840–845.

[42] FISHMAN G. S.: *Concepts and Methods in Discrete Event Digital Simulation*, John Wiley & Sons, 1973.

[43] FJELLHEIM R.: *Cernet Packet Routing Method*, Computer Communications, Vol. 3, Nr. 4, August 1980, Seite 173–176.

[44] FLOYD R. W.: *Algorithm 97: Shortest Path*, Communications of the ACM, Vol. 5, Nr. 6, 1962, Seite 345.

[45] FORD L. R. JR., FULKERSON D. R.: *Flows in Networks*, Princeton University Press, 1962.

[46] FRATTA L., GERLA M., KLEINROCK L.: *The Flow Deviation Method: An Approach to Store-and-Forward Communication Network Design*, Networks, Vol. 3, Nr. 2, 1973, Seite 97–133.

[47] GABRIEL R. P.: *Pattern Languages*, Journal of Object-Oriented Programming, Januar 1994, Seite 72–75.

[48] GAFNI E. M., BERTSEKAS D.: *Asymptotic Optimality of Shortest Path Routing Algorithms*, IEEE Transactions on Information Theory, Vol. 33, Nr. 1, Januar 1987, Seite 83–90.

[49] GALLAGER R.: *A Minimum Delay Routing Algorithm Using Distributed Computation*, IEEE Transactions on Communications, Vol. 25, Nr. 1, Januar 1977, Seite 73–85.

[50] GAMMA E. ET. AL.: *Design Patterns, Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[51] GARCIA-LUNA-ACEVES J. J.: *A Distributed, Loop-Free, Shortest-Path Routing Algorithm*, Proceedings of the IEEE Infocom'88, New Orleans LO, 1988, Paper 11C.3.

[52] GARCIA-LUNA-ACEVES J. J.: *A Minimum-hop routing Algorithm Based on Distributed Information*, Computer Networks and ISDN Systems, Vol. 16, 1988/89, Seite 367–382.

[53] GARCIA-LUNA-ACEVES J. J.: *A Unified Approach to Loop-Free Routing Using Distance Vectors or Link States*, ACM Computer Communication Review, Vol. 19, Nr. 4, September 1989, Seite 212–223.

[54] GAUTHIER P., CHEMOUIL P.: *A System for Testing Adaptive Routing in France*, Proceedings of Globecom'87, Tokyo, Japan, 1987, Paper 23.4.1.

[55] GAVISH B., HANTLER S. L.: *An Algorithm for Optimal Route Selection in SNA Networks*, IEEE Transactions on Communications, Vol. 31, Nr. 10, Oktober 1983, Seite 1154–1161.

[56] GAVISH B., NEUMAN I.: *A System for Routing and Capacity Assignment in Computer Communication Networks*, IEEE Transactions on Communications, Vol. 37, Nr. 4, April 1989, Seite 360–366.

[57] GAVISH B., NEUMAN I.: *Routing in a Network with Unreliable Components*, IEEE Transactions on Communications, Vol. 40, Nr. 7, Juli 1992, Seite 1248–1258.

[58] GLAZER D. W., TROPPER C.: *A New Metric for Dynamic Routing Algorithms*, IEEE Transactions on Communications, Vol. 38, Nr. 3, März 1990, Seite 360–367.

[59] GORA W.: *Who is Who in der Kommunikationswelt*, Datacom, Vol. 7, Nr. 10, Oktober 1990, Seite 132–142.

[60] HÄNDEL R., HUBER M. N., SCHRÖDER S.: *ATM Networks, Concepts, Protocols, Applications*, Second Edition, Addison-Wesley, 1994.

[61] HEDRICK C.: *Routing Information Protocol*, RFC 1058, Juni 1988.

[62] HOPPER A., WHEELER D. J.: *Binary Routing Networks*, IEEE Transactions on Computers, Vol. 28, Nr. 10, Oktober 1979, Seite 699–703.

[63] HSIEH W., GITMAN I.: *Routing Strategies in Computer Networks*, IEEE Computer, Vol. 17, Nr. 6, Juni 1984, Seite 46–56.

[64] HUANG H. K.; SUDA T.: *Collision Avoidance Tree Networks*, Computer Networks and ISDN-Systems, Vol. 26, 1994, Seite 895–911.

[65] HUBER M. N., ET. AL.: *Performance Modelling of a Highly Modularized Packet Switching Node*, Proceedings of the 8th ICCC, München, 1986, Seite 600–605.

[66] HUBER M. N.: *Ein Netzknotenkonzept für integrierte Durchschalte- und Paketvermittlung*, 49. Bericht über verkehrstheoretische Arbeiten, Dissertation, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1990.

[67] HUMBLET P. A., SOLOWAY S. R.: *Topology Broadcast Algorithms*, Computer Networks and ISDN Systems, Vol. 16, 1988/89, Seite 179–186.

[68] HUMBLET P. A.: *Another Adaptive Distributed Shortest Path Algorithm*, IEEE Transactions on Communications, Vol. 39, Nr. 6, Juni 1991, Seite 995–1003.

[69] ISO 7498: *Information Processing Systems — Open Systems Interconnection — Basic Reference Model*, International Standard, 1984.

[70] ISO 8208: *Information Processing Systems — Data Communication — X.25 Packet level Protocol for Data Terminal Equipment*, International Standard, 1990.

[71] ISO 8348: *Information Processing Systems — Open Systems Interconnection — Network Service Definition*, International Standard, 1993.

[72] ISO 8473-1: *Information Technology — Protocol for Providing the Connectionless-mode Network Service*, International Standard, 1994.

[73] ISO 8648: *Information Processing Systems — Open Systems Interconnection — Internal Organization of the Network Layer*, International Standard, 1988.

[74] ISO 8802-3: *Information Processing Systems — Data Communications — Local Area Networks — Part 3: CSMA/CD Access Method and Physical Layer Specifications*. International Standard, 1993.

[75] ISO 8802-4: *Information Processing Systems — Data Communications — Local Area Networks — Part 4: Token-Passing Bus Access Method and Physical Layer Specifications*, International Standard, 1990.

[76] ISO 8802-5: *Information Processing Systems — Data Communications — Local Area Networks — Part 5: Token Ring Access Method and Physical Layer Specifications*, International Standard, 1992.

[77] ISO 9314-2: *Fiber Distributed Data Interface (FDDI) — Token Ring Media Access Control*, International Standard, 1989.

[78] ITG-Empfehlung 1.2-01: *Architekturen und Verfahren der Vermittlungstechnik*, Begriffswerk des ITG-Fachausschusses 1.2, 1995.

[79] ITU-T Recommendation X.25: *Interface Between Data Terminal Equipment (DTE) and Data Circuit-Terminating Equipment (DCE) for Terminals Operating in the Packet Mode and Connected to Public Data Networks by Dedicated Circuit*, 1993.

[80] ITU-T Recommendation X.75: *Packet-Switched Signalling System Between Public Networks Providing Data Transmission Services*, 1993.

[81] Jaffe J., Moss F.H.: *A Responsive Distributed Routing Algorithm for Computer Networks*, IEEE Transactions on Communications, Vol. 30, Nr. 7, Juli 1982, Seite 1758–1762.

[82] Kelly F. P.: *Network Routing*, Statistical Laboratory, University of Cambridge, Cambridge UK, 1991.

[83] Key P. B., Cope G. A.: *Distributed Dynamic Routing Schemes*, IEEE Communications Magazine, Vol. 28, Nr. 10, Oktober 1990, Seite 54–64.

[84] Khanna A., Zinky J.: *The Revised ARPANET Routing Metric*, ACM Computer Communication Review, Vol. 19, Nr. 4, September 1989, Seite 45–56.

[85] Kleinrock L.: *Queueing Systems*, Vol. 1 Theory, John Wiley & Sons, 1975.

[86] Kleinrock L., Kamoun F.: *Hierarchical Routing for Large Networks*, Computer Networks, Vol. 1, 1977, Seite 155–174.

[87] Kocher H.: *Entwurf und Implementierung einer Simulationsbibliothek unter Anwendung objektorientierter Methoden*, 52. Bericht über verkehrstheoretische Arbeiten, Dissertation, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1994.

[88] Kocher H., Lang M.: *An Object-Oriented Library for Simulation of Complex Hierarchical Systems*, Proceedings of the Object Oriented Simulation Conference OOS'94, Tempe AZ, 1994.

[89] KOLB C.: *Untersuchung optimaler Routing-Strategien*, Diplomarbeit Nr. 1161, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1992.

[90] KRISHNAN K. R.: *Network Control with State-Dependent Routing*, Proceedings of the ITC Specialist Seminar, Adelaide, Australien, 1989.

[91] KRISHNAN K. R.: *Markov Decision Algorithms for Dynamic Routing*, IEEE Communications Magazine, Vol. 28, Nr. 10, Oktober 1990, Seite 66–69.

[92] KRÜGER S.: *Simulation, Grundlagen, Techniken, Anwendungen*, de Gruyter, 1975.

[93] LAI W. S.: *Packet Forwarding*, IEEE Communications Magazine, Vol. 26, Nr. 7, Juli 1988, Seite 8–17.

[94] LANG M.: *Verkehrslenkung in Paketvermittlungsnetzen*, Umdruck zum Hochschulkolleg Rechnerkommunikation, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1991.

[95] LIN Y., YEE J. R.: *Three Algorithms for Routing and Flow Control in Virtual Circuit Networks*, Proceedings of IEEE Globecom'90, San Diego CA, 1990, Seite 339–343.

[96] LUENBERGER D. G.: *Linear and Nonlinear Programming*, Addison-Wesley, 1984.

[97] MAIR E., HAUSMANN H., NAESSL R.: *EWSP— A High Performance Packet Switching System*, Proceedings of the 8th ICCC, München, 1986, Seite 359–364.

[98] DE MARCO T.: *Structured Analysis and System Specification*, Prentice-Hall, 1979.

[99] MAXEMCHUCK N. F.: *Routing in the Manhattan Street Networks*, IEEE Transactions on Communications, Vol. 35, Nr. 5, Mai 1987, Seite 503–512.

[100] MAXEMCHUCK N. F., EL ZARKI M.: *Routing and Flow Control in High-Speed Wide-Area Networks*, Proceedings of the IEEE, Vol. 78, Nr. 1, Januar 1990, Seite 204–221.

[101] MC QUILLAN J. M.: *Adaptive Routing Algorithms for Distributed Computer Networks*, Report AD-781467, National Technical Information Service, U.S. Department of Commerce, 1974.

[102] MC QUILLAN J. M., WALDEN D. C.: *The ARPA Network Design Decisions*, Computer Networks, Vol. 1, 1976/77, Seite 243–289.

[103] MC QUILLAN J. M., FALK G., RICHER I.: *A Review of the Development and Performance of the ARPANET Routing Algorithm*, IEEE Transactions on Communications, Vol. 26, Nr. 12, Dezember 1978, Seite 1802–1811.

[104] MC QUILLAN J. M., RICHER I., ROSEN E.: *The New Routing Algorithm for the ARPANET*, IEEE Transactions on Communications, Vol. 28, Nr. 5, Mai 1980, Seite 711–719.

[105] MERLIN P., SEGALL A.: *A Failsafe Distributed Routing Protocol*, IEEE Transactions on Communications, Vol. 27, Nr. 9, September 1979, Seite 1280–1287.

[106] MEYER B.: *Object-Oriented Software Construction*, Prentice Hall, 1988.

[107] MEYER B.: *Applying "Design by Contrac"*, IEEE Computer, Vol. 25, Nr. 10, Oktober 1992, Seite 40–51.

[108] MINIEKA E.: *Optimization Algorithms for Networks and Graphs*, Dekker, 1978.

[109] MOHANTY B. P., CASSANDRAS C. G., TOWSLEY D.: *Performance Comparison of Routing Algorithms in Packet Switched Networks*, Proceedings of IEEE Globecom'90, San Diego CA, 1990, Seite 327–331.

[110] NELSON D. J., SAYOOD K., CHANG H.: *An Extended Least-Hop Distributed Routing Algorithm*, IEEE Transactions on Communications, Vol. 38, Nr. 4, April 1990, Seite 520–528.

[111] PAGE B.: *Diskrete Simulation, Eine Einführung mit Modula-2*, Springer, 1991.

[112] PAWLIKOWSKI K.: *Steady-State Simulation of Queueing Processes: A Survey of Problems and Solutions*, ACM Computing Surveys, Vol. 22, Nr. 2, Juni 1990, Seite 123–170.

[113] PERLMAN R.: *Fault Tolerant Broadcast of Routing Information*, Computer Networks, Vol. 7, 1983, Seite 395–405.

[114] DE PRYCKER M.: *Asynchronous Transfer Mode, Solution for Broadband ISDN*, Ellis Horwood, 1991.

[115] QUATERMAN J., HOSKINS J.: *Notable Computer Networks*, Communications of the ACM, Vol. 29, Nr. 10, Oktober 1986, Seite 932–971.

[116] RAJAGOPALAN B., FAIMAN M.: *A New Responsive Distributed Shortest-Path Routing Algorithm*, ACM Computer Communication Review, Vol. 19, Nr. 4, September 1989, Seite 237–246.

[117] RAJAMARAN A.: *Routing in TYMNET*, Proceedings of European Computing Conference, London, England, 1978, Seite 9–21.

[118] REGNIER J., CAMERON W. H.: *State-Dependent Dynamic Traffic Management for Telephone Networks*, IEEE Communications Magazine, Vol. 28, Nr. 10, Oktober 1990, Seite 42–53.

[119] ROSEN E.: *The Updating Protocol of ARPANET's New Routing Algorithm*, Computer Networks, Vol. 4, 1980, Seite 11–19.

[120] RUDIN H.: *On Routing and "Delta Routing": A Taxonomy and Performance Comparison of Techniques for Packet-Switched Networks*, IEEE Transactions on Communications, Vol. 24, Nr. 1, Januar 1976, Seite 43–59.

[121] SALTI M., BAKRY S. H., AL-TURAIGI M.: *Simulation of Congestion Avoidance and Routing in Packet Switching Networks*, Computer Communications, Vol. 14, Nr. 10, Dezember 1991, Seite 608–615.

[122] SASAKI G., HAJEK B.: *Optimal Dynamic Routing in Single Commodity Networks by Iterative Methods*, IEEE Transactions on Communications, Vol. 35, Nr. 11, November 1987, Seite 1199–1206.

[123] SCHNEIDER M.: *Untersuchungen zu Shortest-Path Algorithmen*, 1. Semesterarbeit Nr. 1280, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1994.

[124] SCHWARTZ M., CHEUNG C. K.: *The Gradient Projection Algorithm for Multiple Routing in Message-Switched Networks*, IEEE Transactions on Communications, Vol. 24, Nr. 4, April 1976, Seite 449–456.

[125] SCHWARTZ M., STERN T. E.: *Routing Techniques Used in Computer Communication Networks*, IEEE Transactions on Communications, Vol. 28, Nr. 4, April 1980, Seite 539–552.

[126] SCHWARTZ M.: *Telecommunication Networks: Protocols, Modeling and Analysis*, Addison-Wesley, 1987.

[127] SEGALL A.: *The Modeling of Adaptive Routing in Data-Communication Networks*, IEEE Transactions on Communications, Vol. 25, Nr. 1, Januar 1977, Seite 85–95.

[128] SEGALL A.: *Optimal Distributed Routing for Virtual Line-Switched Data Networks*, IEEE Transactions on Communications, Vol. 27, Nr. 1, Januar 1979, Seite 201–209.

[129] SEGALL A.: *Advances in Verifiable Fail-Safe Routing Procedures*, IEEE Transactions on Communications, Vol. 29, Nr. 4, April 1981, Seite 491–497.

[130] SEGALL A., SIDI M.: *A Failsafe Distributed Protocol for Minimum Delay Routing*, IEEE Transactions on Communications, Vol. 29, Nr. 5, Mai 1981, Seite 689–695.

[131] SEGALL A.: *Distributed Network Protocols*, IEEE Transactions on Information Theory, Vol. 29, Nr. 1, 1983.

[132] SEGALL A., JAFFE J. M.: *Route Setup with Local Identifiers*, IEEE Transactions on Communications, Vol. 34, Nr. 1, Januar 1986, Seite 45–53.

[133] SHANKAR A. U., ET. AL.: *Performance Comparison of Routing Protocols under Dynamic and Static File Transfer Connections*, ACM Computer Communication Review, Vol. 22, Nr. 5, Oktober 1992, Seite 39–52.

[134] SHANKAR A. U., ET. AL.: *Performance Comparison of Routing Protocols using MaRS: Distance-Vector versus Link-State*, Performance Evaluation Review, Vol. 20, Nr. 1, Juni 1992, Seite 181–192.

[135] SHIER D. R.: *On Algorithms for Finding k Shortest Paths in a Network*, Networks, Vol. 9, Nr. 3, 1979, Seite 195–214.

[136] SHIN K., CHEN M.: *Minimal Order Loop-Free Routing Strategy*, IEEE Transactions on Computers, Vol. 39, Nr. 7, Juli 1990, Seite 870–881.

[137] SIDHU D., NAIR R., ABDALLAH S.: *Finding Disjoint Paths in Networks*, ACM Computer Communication Review, Vol. 21, Nr. 4, September 1991, Seite 43–51.

[138] SIDHU D., ABDALLAH S., NAIR R.: *Congestion Control in High Speed Networks via Alternate Path Routing*, Journal of High Speed Networks, Vol. 2, 1992, Seite 129–144.

[139] SIMHA R., NARAHARI B.: *Single Path Routing with Delay Considerations*, Computer Networks and ISDN Systems, Vol. 24, 1992, Seite 405–419.

[140] SOLOWAY S. R., HUMBLET P. A.: *Distributed Network Protocols for Changing Topologies: A Counterexample*, IEEE Transactions on Communications, Vol. 39, Nr. 3, März 1991, Seite 360–361.

[141] SPINELLI J., GALLAGER R. G.: *Event Driven Topology Broadcast without Sequence Numbers*, IEEE Transactions on Communications, Vol. 37, Nr. 5, Mai 1989, Seite 468–474.

[142] SPROULE D., MELLOR F.: *Routing, Flow and Congestion Control in the Datapac Network*, IEEE Transactions on Communications, Vol. 29, Nr. 4, April 1981, Seite 386–391.

[143] STACEY R. R., SONHURST D. J.: *Dynamic Alternative Routing in the British Telecom Trunk Network*, Proceedings of ISS'87, Phoenix AZ, 1987.

[144] STOKES A. V.: *OSI Standards and Acronyms*, Second Edition, Blenheim Online Publications,1988.

[145] TANENBAUM A. S.: *Computer Networks*, Second Edition, Prentice-Hall International, 1989.

[146] TODE H., ET. AL.: *Traffic Distributing Algorithm for Multicast Routing in Packet Type Networks*, IEICE Transactions, Vol. 74, Nr. 12, Dezember 1991, Seite 4051–4059.

[147] TOPKIS D. M.: *A k Shortest Path Algorithm for Adaptive Routing in Communications Networks*, IEEE Transactions on Communications, Vol. 36, Nr. 7, Juli 1988.

[148] TSITSIKLIS J. N., BERTSEKAS D.: *Distributed Asynchronous Optimal Routing in Data Networks*, IEEE Transactions on Automatic Control, Vol. 31, Nr. 4, April 1986, Seite 325–332.

[149] TURNER J. S.: *Design of an Integrated Services Packet Network*, Proceedings of the 9th Data Communications Symposium, Vancouver, Kanada, 1985, Seite 124–133.

[150] TYMES L.: *Routing and Flow Control in TYMNET*, IEEE Transactions on Communications, Vol. 29, Nr. 4, April 1981, Seite 392–398.

[151] VAZQUEZ E., FERNANDEZ D., VINYES J.: *Packet Switched Network Simulator Based on Modula-2*, Proceedings of the 4. International Conference on Data Communication Systems and their Performance, Barcelona, Spanien, 1990, Seite 409–426.

[152] WÄLDE K.: *Analytische Untersuchung von Multipath-Algorithmen*, 1. Semesterarbeit Nr. 1283, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1994.

[153] WANG Z., CROWCROFT J.: *Shortest Path First with Emergency Exits*, ACM Computer Communication Review, Vol. 20, Nr. 4, September 1990, Seite 166–176.

[154] WANG Z., BROWNING D. W.: *An Optimal Distributed Routing Algorithm*, IEEE Transactions on Communications, Vol. 39, Nr. 9, September 1991, Seite 1379–1388.

[155] WANG Z., CROWCROFT J.: *Analysis of Shortest-Path Routing Algorithms in a Dynamic Network Environment*, ACM Computer Communication Review, Vol. 22, Nr. 2, April 1992, Seite 63–71.

[156] WATANABE Y., ODA T.: *Dynamic Routing Schemes for International Networks*, IEEE Communications Magazine, Vol. 28, Nr. 10, Oktober 1990, Seite 70–75.

[157] WEINTRAUB A.: *The Shortest and k-Shortest Routes as Assignment Problems*, Networks, Vol. 3, Nr. 1, 1973, Seite 61–73.

[158] WILE G. E., GOWAN D. S.: *The Architecture of the DPN Networking System*, Proceedings of the 8th ICCC, München, 1986, Seite 365–369.

[159] WONG J. S., KANG Y.: *Distributed and Fail-Safe Routing Algorithms in Toroidal-Based Metropolitan Area Networks*, Computer Networks and ISDN Systems, Vol. 18, 1989/90, Seite 379–391.

[160] YAMAMOTO H., ET. AL.: *Dynamic Routing Schemes for Advanced Network Management*, IEICE Transactions, Vol. 74, Nr. 12, Dezember 1991, Seite 3981–3991.

[161] YEN J. Y.: *Finding the k Shortest Loopless Paths in a Network*, Management Science, Vol. 17, Nr. 11, Part A: Theory Juli 1971, Seite 712–716.

[162] YOURDON E., CONSTANTINE L.: *Structured Design*, Prentice-Hall, 1979.

[163] ZAUMEN T. W., GARCIA-LUNA ACEVES J. J.: *Dynamics of Distributed Shortest-Path Routing Algorithms*, ACM Computer Communication Review, Vol. 21, Nr. 4, September 1991, Seite 31–42.

[164] ZHANG Z., HARTMANN H. L.: *Optimal Routing in Virtual Circuit Communication Networks*, Proceedings of the ITC-13, Copenhagen, Dänemark, 1991, Seite 395–400.

[165] ZHANG Z., HARTMANN H. L.: *On the Nonbifurcated Routing in Virtual Circuit Communication Networks*, European Transactions on Telecommunications, Vol. 3, Nr. 1, Januar/Februar 1992, Seite 45–53.

[166] ZIMMERMANN U.: *Linear and Combinatorial Optimization in Ordered Algebraic Structures*, Anals of Discrete Mathematics 10,The Netherlands: North Holland, 1981.

# Anhang A

# Notation für OOD-Diagramme

Zur Veranschaulichung der Softwarekonzepte werden in dieser Arbeit häufig Klassen- und Objektdiagramme verwendet. Die verwendete Darstellung orientiert sich an der Booch-Notation, die von Grady Booch entwickelt wurde. Die vollständige Notation umfaßt mehrere Diagrammtypen und ist weit umfangreicher als sie hier eingesetzt wird. In den nachfolgenden Abschnitten sollen nur die wichtigsten Elemente erläutert werden, die für das Verständnis der Arbeit notwendig sind. Eine vollständige Darstellung ist in [20] enthalten.

## A.1 Klassendiagramme

Klassendiagramme beschreiben die Beziehungen zwischen Klassen. Eine Klasse wird durch eine gestrichelte Wolke dargestellt, in deren Mitte der Name der Klasse steht. Außer dem Klassennamen können weitere Elemente innerhalb der Wolke angegeben werden, die durch eine horizontale Linie vom Klassennamen abgetrennt sind. Dies können z.B. die wesentlichen Felder oder Methoden einer Klasse sein. Die Anzahl der Instanzen, die eine Klasse besitzen darf, kann innerhalb geschweifter Klammern angegeben werden. In der Arbeit wurde dies hauptsächlich zur Kennzeichnung von globalen Objekten verwendet. Die zugehörige Klasse besitzt in diesem Fall genau eine Instanz. Abstrakte Klassen können keine Instanzen besitzen. Dies kann entweder durch eine in geschweiften Klammern stehende Null oder durch ein A (für abstrakt) in einem auf dem Kopf stehenden Dreieck dargestellt werden.

Die wichtigsten Beziehungen zwischen Klassen sind Vererbung, Benutzen anderer Klassen und das Enthalten anderer Klassen. Die Vererbung wird durch einen Pfeil dargestellt, der von der abgeleiteten Klasse zur Basisklasse gerichtet ist. Für eine Benutzen-Beziehung wird eine Linie mit einem nicht ausgefüllten Kreis verwendet. Der Kreis ist dabei bei der Klasse, die die Dienste der anderen Klasse in Anspruch nimmt. Die Enthalten-Beziehung

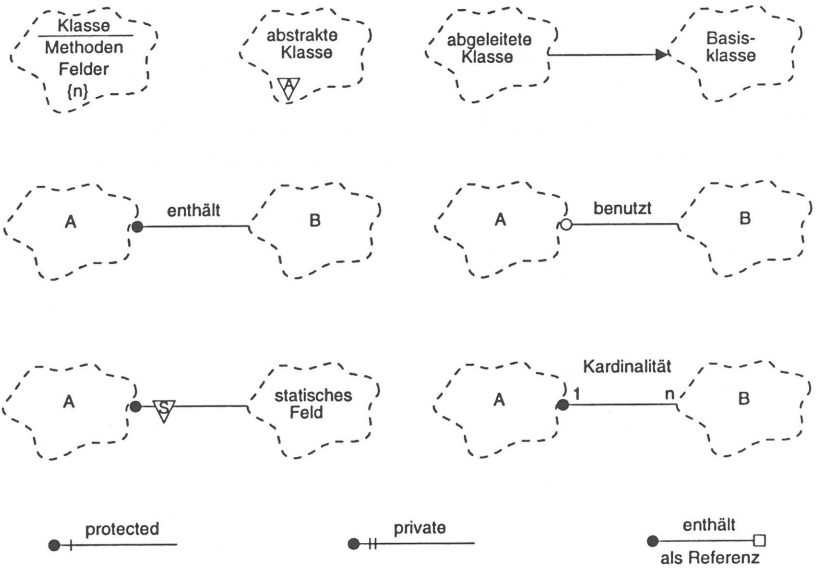



Bild A.1: Notation für Klassendiagramme

wird durch eine Linie mit einem ausgefüllten Kreis dargestellt. Auch hier ist der Kreis bei der Klasse, die Instanzen der anderen Klasse enthält. Ist die Anzahl der Objekte bekannt, die an einer Beziehung beteiligt sind, kann dies durch Angabe der entsprechenden Zahlen an den Enden der Beziehungslinie verdeutlicht werden. Als Beispiel verwalte ein Objekt der Klasse A viele Instanzen der Klasse B. An dem Ende der Beziehungslinie, das der Klasse A zugewandt ist, steht eine 1. Am anderen Ende der Beziehung, also in der Nähe der Klasse B, gibt ein $n$ an, daß die Klasse A mehrere Objekte der Klasse B verwaltet. Zugriffsrechte werden durch kurze Striche durch die Beziehungslinie dargestellt. Ein Strich steht für `protected`, zwei Striche für `private`. Enthält eine Klasse ein Objekt einer anderen Klasse nicht direkt, sondern eine Referenz auf dieses Objekt, kann dies durch ein weißes Quadrat am Ende der Enthaltensein-Beziehung angegeben werden. Ist ein Objekt der Klasse B nicht in jeder Instanz der Klasse A vorhanden, sondern existiert für alle Instanzen von A nur ein gemeinsames Objekt der Klasse B, bezeichnet man dieses auch als statisches Feld der Klasse A. Dies kann durch ein S (für statisch) in einem kleinen Dreieck dargestellt werden, das bei der zugehörigen Enthaltensein-Beziehung steht.

In Abbildung A.1 sind die beschriebenen Elemente der Notation nochmals gemeinsam dargestellt.

## A.2 Objektdiagramme

Objektdiagramme haben im wesentlichen zwei Aufgaben: Mit ihrer Hilfe kann zum einen die Struktur der Objekte zur Laufzeit eines Programms ausgedrückt werden, zum anderen lassen sich mit ihnen dynamische Abläufe des Programms darstellen. Die wesentlichen Elemente, die in Objektdiagrammen verwendet werden, sind in Abbildung A.2 dargestellt.

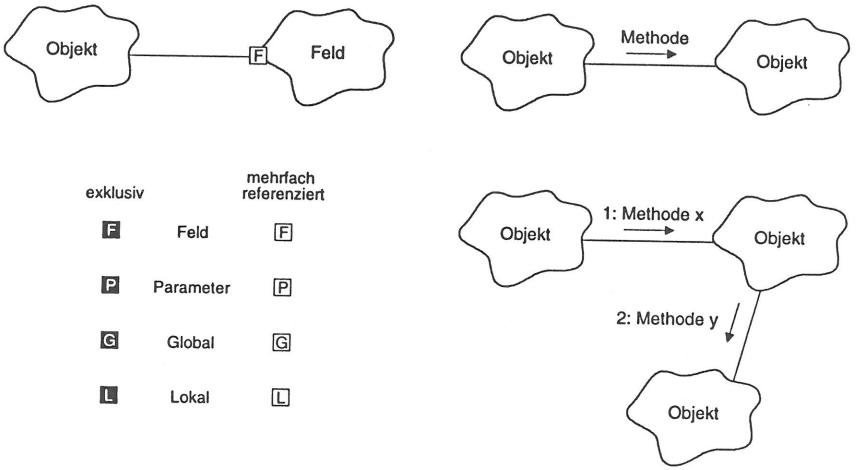



Bild A.2: Elemente von Objektdiagrammen

Beziehungen zwischen Objekten werden durch Linien gekennzeichnet. Die Sichtbarkeit der Objekte kann durch die Angabe von sogenannten *Adornments* näher bestimmt werden. Adornments sind kleine Quadrate mit Buchstaben in der Mitte, die am Ende einer Beziehung stehen können. Ein „F“ drückt dabei aus, daß ein Objekt als Feld in einem anderen Objekt enthalten ist, ein „P“ bedeutet, daß das Objekt Parameter einer Methode des anderen ist, ein „G“ steht für ein globales Objekt und ein „L“ bedeutet, daß es sich um ein lokales Objekt handelt. Ist das Quadrat am Ende der Beziehung ausgefüllt, kann das eine Objekt exklusiv dem anderen zugeordnet werden. Ist das Quadrat dagegen nicht ausgefüllt, wird das Objekt gleichzeitig von mehreren Objekten referenziert. Am deutlichsten wird dieser Unterschied am Beispiel des Feldes. Ein ausgefülltes Quadrat mit einem „F“ in der Mitte bedeutet demnach, daß das Objekt, bei dem das Quadrat steht, exklusiv als Feld in dem anderen Objekt enthalten ist. Ein nicht ausgefüllter Kasten mit einem „F“ bedeutet dagegen, daß das eine Objekt nur einen Zeiger auf das andere besitzt.

Mit Hilfe von Objektdiagrammen lassen sich auch dynamische Abläufe darstellen. Ruft ein Objekt eine Methode eines anderen Objekts auf, wird dies durch einen Pfeil mit

dem Methodenname über der Beziehung dargestellt. Werden mehrere Methoden aufgerufen, kann die Reihenfolge der Aufrufe durch Nummern vor dem Methodenname deutlich gemacht werden.