Lehrstuhl für Kommunikationsnetze
Technische Universität München

Fehlertolerante Overlay-Netze mit programmierbaren Netzknoten

Christian Bachmeir

Vollständiger Abdruck der von der Fakultät für
Elektrotechnik und Informationstechnik der Technischen Universität München
zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs

genehmigten Dissertation.

Vorsitzender:		Univ.-Prof. Dr.-Ing. Dr.-Ing. h.c. Dierk Schröder
Prüfer der Dissertation:	1.	Univ.-Prof. Dr.-Ing. Jörg Eberspächer
	2.	Univ.-Prof. Dr. sc. techn. ETH Andreas Herkersdorf

Die Dissertation wurde am 01.07.2004 bei
der Technischen Universität München
eingereicht und durch die
Fakultät für Elektrotechnik und Informationstechnik
am 24.05.2005 angenommen.

Vorwort

Diese Arbeit ist im Laufe meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Kommunikationsnetze der Technischen Universität München entstanden.

Meinen besonderen Dank spreche ich meinem Doktorvater, Herrn Prof. Dr.-Ing. Jörg Eberspächer dafür aus, dass er mir die Möglichkeit gegeben hat, in einem dreijährigen, standortübergreifenden Projekt intensiv zu forschen.
Darüber hinaus durfte ich herausfordernde Aufgaben in der Organisation und Lehre als Affiliated-Assistant im Center for Digital Technology & Management (CDTM) der Technischen Universität München und der Ludwig-Maximilians-Universität München wahrnehmen.

Außerdem bedanke ich mich bei Herrn Prof. Dr. sc. techn. Andreas Herkersdorf für die Übernahme des Zweitgutachtens.

Ein herzliches Dankeschön geht ebenfalls an meine Kolleginnen und Kollegen am Lehrstuhl. Der konstruktive wissenschaftliche Austausch, die zahlreichen Hilfestellungen, und nicht zuletzt das hervorragende Arbeitsklima waren sicher eine gute Basis für das Gelingen dieser Arbeit.
Mein Dank gilt insbesondere Herrn Dipl.-Ing. Peter Tabery, meinem Bürokollegen und Projektmitstreiter. Gemeinsam haben wir einige wissenschaftliche Untiefen erfolgreich umschifft.

An dieser Stelle möchte ich mich auch bei meinen Studenten für deren unermüdlichen Einsatz beim Bau der Prototypen bedanken.

München, im Juli 2004

Christian Bachmeir

Kurzfassung

Diese Arbeit führt das Konzept der transparenten Fehlertoleranz als Dienst in Programmierbaren Netzen ein. Die Hauptbeiträge der Arbeit sind die Architektur eines adaptiven Fehlertoleranzdienstes und eine fehlertolerante, komponentenbasierte Architektur der zugrundeliegenden programmierbaren Netzknoten.

Der Fehlertoleranzdienst basiert auf einem integrierten, mehrdimensionalen Ansatz. Ziel ist die Bereitstellung von domänenübergreifender, applikationsspezifischer Fehlertoleranz im Internet. Der Dienst baut auf der Internetschicht auf und maskiert Applikationsfehler durch Eingriffe in die höheren Schichten.
Die Arbeit integriert das Konzept der longitudinalen, horizontalen und vertikalen Fehlermaskierung bzw. Fehlerabschwächung. Dies geschieht durch die Bereitstellung von disjunkten Pfaden bei der Datenübertragung, die Integration von kooperierenden, innovativen Cacheing-Mechanismen, die Einbindung von Content-Distribution-Netzen und die direkte, adaptive Unterstützung der zu schützenden Applikationen.
Wesentliches Merkmal der Dienstarchitektur ist die topologische Trennung der Signalisierungsfunktionen von der eigentlichen Bereitstellung der Fehlertoleranz bzw. der Fehlermaskierung. Die Signalisierung realisieren wir durch skalierbare, dedizierte Overlay-Netze. Durch die Kooperation dieser Overlay-Netze werden disjunkte Pfade bzw. alternativer Content im Netz gefunden.
Die Fehlermaskierung erfolgt auf Instanzen von Applikationen zugeschnitten. Somit ist eine flow-basierte, vertikale Fehlermaskierung als direkte Unterstützung der Applikationen möglich.
Der Datentransport erfolgt in dynamischen Overlay-Netzen. Diese Arbeit präsentiert einen neuen Mechanismus für den Datentransport in Overlay-Netzen. Durch iterative Manipulation von bereits vorhandenen Feldern in Datenpaketen wird die Effizienz verbessert.

Die vorgeschlagene Knotenarchitektur teilt die Grundfunktionen eines programmierbaren Knotens (*Routing, Steuerung und Bereitstellung von Diensten*) auf dedizierte Komponenten bzw. Maschinen auf.
Wesentliche Neuerung ist hier die Umgehung einer expliziten Kommunikation zwischen Kern und Userspace. Wir schlagen vor, Standard-Router durch ein Enkapsulierungs-/ Dekapsulierungs-Modul zu erweitern. Die eigentliche Bereitstellung von Diensten erfolgt auf beigestellten Maschinen, die ausschließlich auf Standardfunktionen eines Betriebssystems zurückgreifen. Dadurch sind diese Maschinen unabhängig vom verwendeten Betriebssystem.
Ein weiteres Merkmal ist die topologische Trennung der Knotensteuerung von der Bereitstellung der Dienste. Dadurch ist es möglich, ladbare Dienste skalierbar und fehlertolerant anzubieten. Darüber hinaus wird die Sicherheit gegenüber Angriffen verbessert.
Ein weiterer Vorteil der Architektur ist eine mögliche Konvergenz mit Grid-Computing. Die Verwendung von offenen Signalisierungsschnittstellen, in Kombination mit der Unabhängigkeit von Betriebssystemen, ermöglicht es, die Architektur in ein Grid einzubinden. Somit können im Netz vorhandene Ressourcen effizienter genutzt werden.

Sowohl der Fehlertoleranzdienst als auch die Knotenarchitektur wurden prototypisch implementiert und unter realen Bedingungen in Testnetzen bzw. im Internet evaluiert. Dadurch wurde die Realisierbarkeit der Ansätze nachgewiesen.

Kurzfassung für das Jahrbuch der TU-München

Diese Arbeit beschreibt adaptive, mehrdimensionale Fehlertoleranz als Dienst in Programmierbaren Netzen. Wir schlagen eine komponentenbasierte, programmierbare Knotenarchitektur vor. Basierend auf dieser performanten und fehlertoleranten Plattform wird ein skalierbarer Fehlertoleranzdienst eingeführt. Die notwendige Redundanz wird durch kooperierende, auf Netzzwischensystemen aufbauende, transparente Overlay-Netze des Netzdienstes zur Verfügung gestellt. Durch die vorgeschlagene topologische Trennung von Dienstsignalisierung und dynamischer Fehlermaskierung wird Skalierbarkeit erreicht. Der flow-basierte Ansatz erlaubt eine applikationspezifische, adaptive Fehlertoleranz durch den Netzdienst.

Abstract for the yearbook

This thesis describes adaptive multi-dimensional fault tolerance as a network service in Programmable Networks. We outline the concept of a component based programmable node architecture. Based on this performant and fault tolerant plattform we introduce scalable fault tolerance as a transparent network service. The foundation of the service are cooperating transparent overlay networks, located on gateways, which organize the provision of redundancy. Scalabilty of our approach is given through the topological separation of service signaling and dynamic fault masking. Furthermore our flow based approach integrates application specific adaptive fault tolerance in the proposed network service.

Inhaltsverzeichnis

Abkürzungsverzeichnis

AAA	Authentication Admission Accounting
ALMI	Application Level Multicast Infrastructure
ALON	Application Layer Overlay Network
AMnet	Active Multicast Network
ANPE	Active Network Processing Elements
ANTS	Active Node Transfer System
API	Application Programming Interface
APLS	Active Protocol Label Switching
AS	Autonomous System
ATM	Asynchronous Transfer Mode
B2B	Business to Business
B2C	Business to Consumer
BGP	Border Gateway Protocol
CAC	Call Admission Control
CBR	Constant Bit Rate
CDN	Content Distribution-Netze
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
DDoS	Distributed Denial of Service
DNS	Domain Name Service
DoS	Denial of Service
DSL	Digital Subscriber Line
DSP	Digitaler Signalprozessor
DVMRP	Distance Vektor Multicast Protocol
EE	Execution Environment
EWSD	Elektronisches Wählsystem Digital
FASRO	Fast Scoped Rerouting
FEC	Forwarding Error Correction
FPGA	Field Programmable Gatey Array
FT-TCP	Fault Tolerant Transmission Control Protocol
FTP	File Transfer Protocol
GSM	Global System for Mobile Telecommunications
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol over Secure Socket Layer

ICMP	Internet Control Management Protocol
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
IN	Intelligentes Netz
IntServ	Integrated Services Architecture
IP	Internet-Protokoll
ISDN	Integrated Services Digital Network
ISO	International Standards Organization
LAN	Local Area Network
LANE	LAN Emulation
LRZ	Leibniz Rechenzentrum
LSP	Label Switched Path
MBONE	Multicast Backbone Overlay-Netz
MDC	Multiple Description Coding
MPEG	Moving Pictures Experts Group
MPLS	Multiple Protocol Label Switching
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair
MTU	Maximum Transfer Unit
OGSA	Open Grid Service Architecture
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
OTN	Optische Transportnetze
P2P	Peer to Peer
PPD-TCP	Packet Path Diversity Transmission Control Protocol
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RFC	Request For Comment
RON	Resilient Overlay Networks
RPC	Remote Procedure Call
RTT	Round Trip Time
SCTP	Stream Control Transmission Protocol
SDH	Synchrone Digitale Hierarchie
SMTP	Simple Mail Transport Protocol
SNIPLS	Sparse Networkspanning IP Label Switching
ST-TCP	Server Fault Tolerant Transmission Control Protocol
TCP	Transmission Control Protocol
TTL	Time To Live
UDP	User Datagram Protocol
VBR	Variable Bit Rate
VOD	Video on Demand
VoIP	Voice over IP
VPN	Virtual Private Network
WAN	Wide Area Network
WLAN	Wireless Local Area Network
WWW	World Wide Web
XML	eXtensible Markup Language

Kapitel 1

Einführung

Das Internet wurde zunächst in erster Linie für Anwendungen genutzt, die geringe Anforderungen an das Netz stellen, wie z.B. für E-Mail-Kommunikation. Für derartige Anwendungen war die unzuverlässige *best-effort*-Vermittlung von Paketen im Internet ausreichend. In den letzten Jahren begann die Entwicklung des Internets hin zu einer universellen Kommunikationsplattform. Zahlreiche Bestrebungen benutzen mittlerweile das Internet für komplexe Kommunikationsszenarien.
Exemplarisch lassen sich hier z.B. Pop-Konzerte anführen, die live über das Internet übertragen werden [mad00]. Mittlerweile sind derartige Übertragungen sehr beliebt. Die Anzahl der Online-Zuschauer zuhause geht regelmäßig in die Millionen.
Auch Telekonferenzen [SKS00] über das Internet erfreuen sich großer Beliebtheit. Das derzeit diskutierte Voice-over-Internet-Protokoll (VoIP) [voi04] hat u.U. das Potential, den zukünftigen Standard in der Telekommunikationsbranche zu setzen.
Diese beispielhafte Liste für neue Anwendungen die das Internet benutzen, lässt sich fortsetzen. Alle diese neuartigen Anwendungen haben eine Gemeinsamkeit: Die Anforderung an das Netz bzw. an die Netzdienste gehen weit über das gegenwärtig angebotene, reguläre IP-Routing im Internet hinaus.

Dies gilt in besonderem Maße für den Bereich der Fehlertoleranz bei der Kommunikation über das Internet. Die unterschiedlichen Anforderungen der Anwendungen an die Fehlertoleranz können durch Standardverfahren nur eingeschränkt abgedeckt werden.

Diesem Umstand tragen wir in dieser Arbeit Rechnung. Wir schlagen die Bereitstellung von adaptiver Fehlertoleranz als Dienst in Programmierbaren Netzen vor. Dadurch kann die Fehlertoleranz des Netzes dynamisch an spezifische Anforderungen von verteilten Applikationen angepasst werden.

1.1 Motivation

Die Verfügbarkeit von Systemen ist definiert als $\frac{MTTF}{MTTF+MTTR}$*. Hochverfügbare Systeme sind dadurch gekennzeichnet, dass sie pro Jahr nicht mehr als 315 Sekunden unverfügbar sind [Chr02]. Derartige Systeme, wie z.B. das reguläre Telefonnetz, haben somit eine Verfügbarkeit von mindestens 99,999 %.
Die gegenwärtige Verfügbarkeit des Internets, aus Sicht eines Endsystems, ist von diesem Wert noch relativ weit entfernt. In einer kürzlich veröffentlichten Studie [Dam04] wird z.B. die Verfügbarkeit von Servern im Mittel mit 99,3 % angegeben. Der offensichtliche Bedarf an einer Verbesserung der Verfügbarkeit bzw. des Umgangs mit Fehlern ist vor allem durch die Anforderungen von jüngsten Anwendungen gestiegen.

Gegenwärtig gibt es einige Ansätze, die Fehlertoleranz im Internet bereitstellen. Prinzipiell lässt sich zwischen netzbasierten und endsystembasierten Ansätzen unterscheiden.
Netzbasierte Ansätze stellen die Fehlertoleranz entweder in der Internetschicht, oder durch auf die Internetschicht aufgesetzte Overlay-Netze zur Verfügung. Ziel dieser Architekturen ist es, alternative und disjunkte Pfade bereitzustellen. Im Falle eines Ausfalls kann dann mittels Restaurationsverfahren oder Ersatzschaltverfahren auf alternative Wege bei der Datenübertragung zurückgegriffen werden. Diese Ansätze unterstützen somit die zu schützenden Applikationen indirekt.
Endsystembasierte Ansätze berücksichtigen die Applikationen und unterstützen durch fehlertolerante Protokolle u.U. in Kombination mit redundantem oder angepassten Content die Fehlertoleranz der Applikationen direkt.

In dieser Arbeit fokussieren wir uns auf den Fall, dass bei der Datenübertragung durch das Internet kontinuierliche Paketverluste auftreten, die nicht innerhalb von domänenbegrenzten Netzen bzw. seitens der Endsysteme behebbar sind.
Hier helfen Restaurationsverfahren im Border Gateway-Protokoll (BGP). Innerhalb der Internetschicht wird nach einer gewissen Konvergenzzeit wieder ein fehlerfreier Zustand hergestellt. Durch diese *selbstheilende* Charakteristik des Internets ist sichergestellt, dass nach einem Ausfall die Funktionalität mit einer gewissen Verzögerung wieder hergestellt werden kann. Problem in diesem Zusammenhang ist eine möglicherweise lange Dauer der Konvergenzzeit. Für diesen Zeitraum steht in der Regel keine Konnektivität der Kommunkationspartner am Netzrand zur Verfügung.

Aus diesem Grund wird intensiv daran gearbeitet, bestehende Restaurationsverfahren zu verbessern bzw. die Konvergenzzeit zu verringern, um die Fehlertoleranz des Gesamtsystems zu verbessern. Einige Vorschläge verkleinern die Konvergenzzeit erheblich. Die absolute Größenordnung liegt aber auch nach der Verbesserung immer noch in einem Bereich von Minuten. Dies kann von vielen Applikationen nicht toleriert werden.
Deshalb wurden in letzter Zeit Overlay-Netze vorgeschlagen, die Fehler im Netz maskieren. Der Vorteil derartiger Architekturen liegt darin, dass sie ein schnelles Ersatzschalten durch das Overlay-Netz zur Verfügung stellen können. Im Vergleich zu Restaurationsverfahren entfällt somit die Konvergenzzeit.
Die vorgeschlagenen Ansätze sind aber eher für kleine geschlossene Anwendungsszenarien

*MTTF: Mean Time To Failure; MTTR: Mean Time To Repair

konzipiert. Dies liegt in begrenzter Skalierbarkeit und intransparenter Anbindung von Endsystemen begründet.

Ein weiteres nicht zu vernachlässigendes Problem in diesem Zusammenhang ist die eigentliche Bereitstellung der Fehlertoleranz im Netz. Eine sich wandelnde Applikationslandschaft im Internet stellt u.U. sich ständig ändernde, applikationsspezifische Anforderungen an die Fehlertoleranz des Netzes. Der herkömmliche Weg der Standardisierung von Funktionen des Netzes (und die anschließende Umsetzung) erscheint zu träge, um diesen Anforderungen zu genügen.

Der in dieser Arbeit vorgeschlagene adaptive Fehlertoleranzdienst basiert ebenfalls auf Overlay-Netzen. Kooperierende Overlay-Netze werden dabei ausschließlich zur Signalisierung bzw. Konfiguration verwandt. Der Datentransport erfolgt orthogonal zur Signalisierung in einem dedizierten transparenten Overlay-Netz. Dadurch ist es möglich, die Fehlertoleranz netzübergreifend und skalierbar zur Verfügung zu stellen. Darüber hinaus unterstützt der vorgeschlagene Netzdienst Applikationen direkt, durch mehrdimensionale Fehlertoleranz. Des Weiteren schlagen wir in dieser Arbeit vor, den Fehlertoleranzdienst in Programmierbaren Netzen anzubieten. Dadurch ist die Voraussetzung für die notwendige Adaptivität des Netzdienstes gegeben.

1.2 Struktur der Arbeit

Aus Kapitel 2 sind die für diese Arbeit relevanten Grundlagen zu ersehen. Zunächst präsentieren wir die, durch den Aufbau des Internets bedingte, mögliche Pfadredundanz. Des Weiteren gehen wir auf die Entwicklung von Overlay-Netzen ein. Wir stellen das für diese Arbeit wichtige Konzept der Programmierbaren Netze vor. Außerdem geben wir eine kurze Einführung in das Gebiet der Behandlung von Fehlern in Systemen.

Anschließend zeigen wir in Kapitel 3 einen Überblick über gegenwärtige Entwicklungen auf den Gebieten der Fehlertoleranz und der Programmierbaren Netze auf. Wir gehen auf domänenübergreifende Fehlertoleranzmechanismen in der Internetschicht, der Applikationsschicht und in Overlay-Netzen ein. Zusätzlich stellen wir den Stand der Technik bei der Entwicklung von programmierbaren Knotenarchitekturen dar.

In Kapitel 4 geben wir einen Überblick über den Beitrag dieser Arbeit. Wir präsentieren die zugrundeliegenden System- und Fehlermodelle. Darauf aufbauend stellen wir das Konzept der integrierten, mehrdimensionalen Fehlertoleranz vor. Wir schlagen vor, diese als adaptiven Netzdienst in Programmierbaren Netzen bereitzustellen.

Die Architektur des ladbaren Fehlertoleranzdienstes stellen wir in Kapitel 5 im Detail dar. Dabei beschreiben wir im Wesentlichen die Umsetzung der vorgestellten mehrdimensionalen Fehlertoleranz. Der Fehlertoleranzdienst beruht auf der Integration verschiedener kooperierender Overlay-Netze. Durch diese werden Redundanzen im Netz gefunden und zur Fehlermaskierung zur Verfügung gestellt.
Wesentlich ist der flow-basierte Ansatz des Netzdienstes. Somit kann der Dienst auf spezielle Eigenschaften von verteilten Applikationen eingehen und diese direkt unterstützen. Darüber hinaus präsentieren wir ein effizientes Verfahren des Datentransports in Overlay-Netzen.

In Kapitel 6 schlagen wir die Architektur eines komponentenbasierten programmierbaren Knotens vor. Diese Architektur verbessert im Wesentlichen die lokale Skalierbarkeit und die Fehlertoleranz von programmierbaren Architekturen. Dies sind elementare Voraussetzungen für die Bereitstellung des angeführten Fehlertoleranzdienstes.
Der Ansatz basiert auf einer Erweiterung von Standard-Routern zur Router-Komponente. Beigestellte Maschinen bilden eine Rechner-Komponente und stellen die eigentliche Funktionalität eines programmierbaren Knoten zur Verfügung. Dadurch werden die wesentlichen Funktionen eines programmierbaren Knoten (Routing, Steuerung und Bereitstellung von Diensten) topologisch getrennt. Dies erlaubt eine ausschließliche Benutzung von Standardfunktionen bei der Bereitstellung von Diensten.
Zusätzlich verbessert die vorgeschlagene Architektur durch die Verwendung von Standardfunktionen die Sicherheit gegenüber Angriffen, und erlaubt eine Konvergenz mit vorgeschlagenen Architekturen des Grid-Computing.

Kapitel 2

Grundlagen

Dieses Kapitel führt in das Gebiet der vorliegenden Arbeit ein. Ziel ist es, die für die in den nachfolgenden Kapiteln vorgeschlagenen architekturrelevanten Aspekte herauszuarbeiten.
Zunächst wird der Aufbau IP-basierter Netze am Beispiel des Internets dargelegt. Die solide, aber relativ unflexible Architektur stößt bei dem Wunsch neue Netzdienste einzuführen an ihre Grenzen. Dies war einer der Gründe für die Einführung verschiedenster Overlay-Netz-Ansätze.
In diesem Kapitel präsentieren wir einen Überblick über entwickelte Overlay-Netze. Aufsetzend auf anfänglich minimale Vorschläge führte die Entwicklung in den letzten Jahren hin zu komplexen Netzdiensten, die am Netzrand in der Applikationsschicht bereitgestellt werden.

Des Weiteren geben wir eine kurze Einführung in das Gebiet der Programmierbaren Netze. Ziel der vorliegenden Arbeit ist es, einen universellen Fehlertoleranzmechanismus im Netz zur Verfügung zu stellen. Dieser muss unterschiedlichste Anforderungen an die Fehlertoleranz in einem heterogenen Umfeld erfüllen können.
Als Basis des Fehlertoleranzmechanismus sehen wir in dieser Arbeit die Programmierbaren und Aktiven Netze. In diesem relativ jungen Forschungsgebiet wurden bisher einige ladbare Netzdienste vorgeschlagen. Wir fügen – als Teil dieser Arbeit – das Konzept des ladbaren Fehlertoleranzdienstes hinzu.
Zum Schluss dieses Kapitels gehen wir allgemein auf den Fehlerbegriff ein bzw. auf Fehler in Kommunikationssystemen und deren Behandlung.

2.1 IP-basierte Netze

Diese Arbeit beschäftigt sich mit dem Thema Fehlertoleranz in IP-basierten Netzen. In diesem Abschnitt werden die zwei wesentlichen Aspekte IP-basierter Netze, diese Arbeit betreffend, präsentiert.

Zunächst zeigen wir den hierarchielosen Zusammenschluss von Netzen mittels des Internet-Protokolls (IP) am Beispiel des Internets. Das Grundprinzip von IP-basierten Netzen, heterogene und administrativ voneinander unabhängige Domänen hierarchielos miteinander zu

verknüpfen, ist sicherlich einer der Erfolgsfaktoren für die rasante Vergrößerung und Verbreitung des Internets in den letzten Jahren. Mit Hilfe von IP können einzelne Netze mit relativ geringem Aufwand (im Vergleich zu anderen Netzarchitekturen) in skalierbarer Art und Weise zu komplexen Systemen miteinander verbunden werden. Daraus ergeben sich aber nicht nur Vorteile [AK00].
Das Internet (bzw. IP) stellt die Basis dar, um Daten zwischen Rechnern in verschiedenen Netzen transportieren zu können. Dem Endnutzer aber bleibt im Regelfall dieser Mechanismus des Datentransports verborgen. Der Nutzer möchte Dienste (z.B. WWW, E-Mail, Video on Demand, etc.) auf seinem Rechner verwenden. Diese Dienste wiederum nutzen IP, um Informationen zwischen Rechnern im Internet auszutauschen.
Diese Arbeit beschreibt einen mehrdimensionalen, integrierten Ansatz, der Fehler von Diensten gegenüber dem Nutzer bzw. gegenüber Applikationen maskiert. Basis für diese Integration ist die Schichtenarchitektur von IP-Netzen bzw. internetbasierten Applikationen, die in diesem Abschnitt dargelegt wird.

2.1.1 Heterogene, administrative Domänen

In Abbildung 2.1 ist ein exemplarischer Zusammenschluss mehrerer IP-Netze dargestellt. Jedes dieser Netze steht unter eigener Administration und ist von den anderen Netzen unabhängig. Als Beispiel könnte man hier die Vernetzung von Universitäten anführen (Netz 1 - 4), die über das Internet-Protokoll miteinander verbunden sind. Wesentliche Merkmale sind folgende:

- Jedes Netz ist administrativ eigenständig.
- Netze sind über Router (bzw. Gateways) lose miteinander verbunden.
- Das Routing von Datenpaketen erfolgt IP-basiert.

Eine der wesentlichen Eigenschaften der Routingfunktionalität im Internet ist, dass Router, wenn sie ein Paket weiterleiten, die Adresse des Ziel-Netzes benutzen und nicht die eigentliche IP-Adresse des Ziel-Computers [Com00]. In Abbildung 2.1 werden exemplarisch Datenpakete von *Host A* nach *Host B* übertragen. Die Pakete werden über den *Router R* geroutet. In der lokalen *Routingtabelle* des Routers R ist aber nicht die IP-Adresse von Host B eingetragen, sondern nur die Adresse des Ziel-Netzes (in diesem Fall die Adresse des Netzes 3), in welchem sich Host B befindet. Durch diese Vereinfachung ist es möglich, sehr viele unabhängige Netze (bzw. Subnetze) miteinander zu verbinden.
Im Internet wird ein Netz, das unter einer Administration steht und dem ein gewisser Satz an IP-Adressen zugeordnet ist, als ein Autonomes System (AS) bezeichnet. Als Beispiel für ein Autonomes System könnte man z.B. ein Hochschulnetz anführen. Unter zentraler Verwaltung wird in diesem AS die Anbindung an andere Autonome Systeme geregelt, und somit die Verbindung zum Internet gewährleistet.
Innerhalb eines AS kann es zusätzlich noch viele kleine Netze (Sub-Netze) geben, z.B. ein Local Area Network (LAN) eines bestimmten Lehrstuhls. Wir sehen somit, dass ein Autonomes System ein heterogenes, komplexes Netz sein kann. Durch die Zusammenfassung von Netzen in Autonome Systeme in Verbindung mit den Routingmechanismen des IP, wie z.B. BGP und OSPF [Com00], besteht das heutige Internet aus ca. 15.000 AS, die miteinander

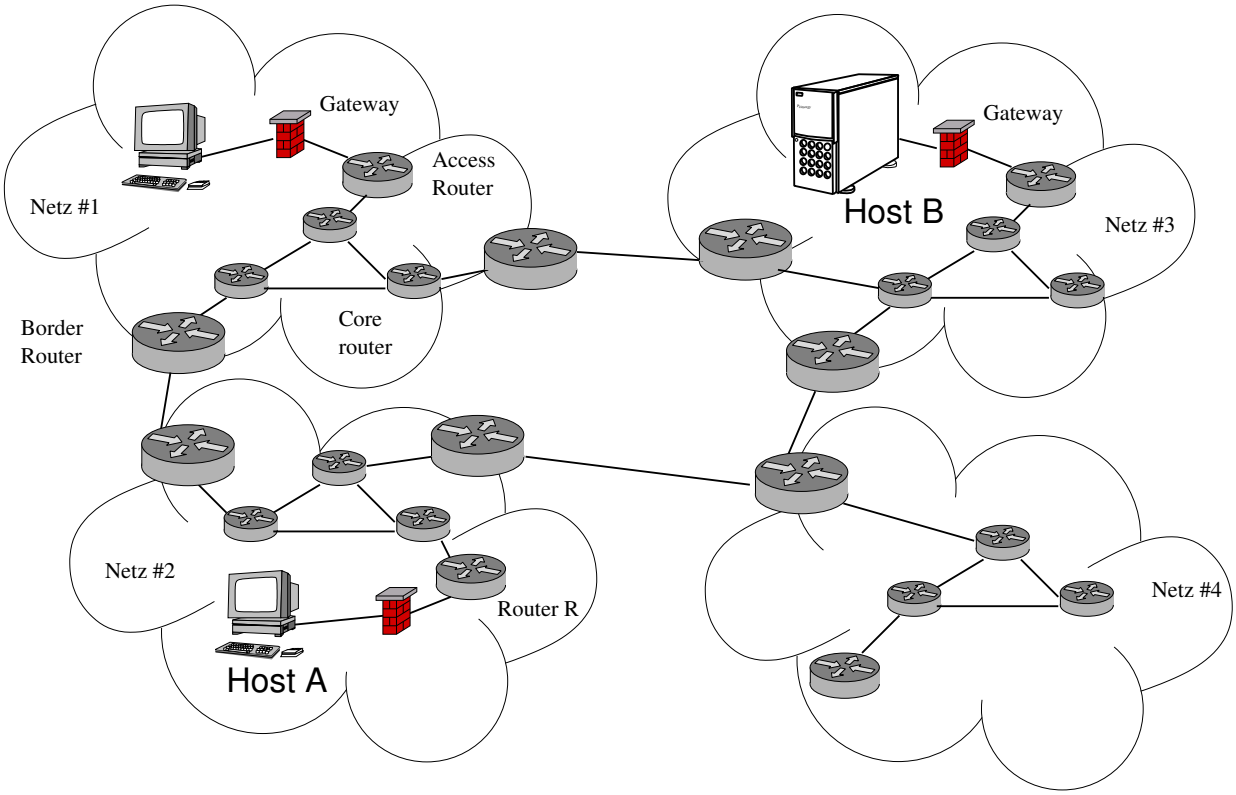

Abbildung 2.1: Aufbau von IP-Netzen

verbunden sind. Insgesamt sind derzeit ca. 200 Millionen Computer [isc03] vernetzt.
Aus praktischen bzw. Kostengründen ist nicht jedes AS mit jedem anderen AS direkt verbunden. Wie in unserem Beispiel exemplarisch dargestellt, können Autonome Systeme auch dazu benutzt werden, Datenpakete von einem AS in ein anderes AS weiterzuleiten. Mehrere Autonome Systeme können in einer Kette zur Weiterleitung benutzt werden. Die Organisation des Datentransports übernehmen dann Interdomain Routing-Protokolle, wie z.B. BGP. **Dieses Grundprinzip führt zu einer wesentlichen Voraussetzung für diese Arbeit: Durch den Aufbau des Internets besteht die Möglichkeit, für bestimmte Kommunikationsverbindungen einen alternativen Datenpfad durch das Netz zu finden. In Abbildung 2.1 ist dies exemplarisch veranschaulicht. Datenpakete einer Kommunikationsverbindung zwischen Host A und Host B können z.B. über *Netz #4* oder über *Netz #1* geroutet werden.**
In unserem Beispiel ist es somit möglich im Falle des Ausfalls eines Datenpfades (z.B. in Netz #1), Datenpakete über einen alternativen Pfad (Netz #4) zu routen. Dieses Grundprinzip des Internets nennt man Wegeredundanz.

2.1.2 Schichtenarchitektur

Der vorhergehende Abschnitt geht auf die Zusammensetzung von IP-Netzen (bzw. des Internets) aus vielen gleichberechtigten Subnetzen ein. Im Folgenden gehen wir auf die Funktionsweise der Kommunikation, von Computern bzw. Applikationen, die über das Internet verbunden sind, ein.
Exemplarisch nehmen wir einen vereinfachten Fall, dargestellt in Abbildung 2.2, an: Zwei Netze (z.B. Local Area Networks) sind über einen *Router R* miteinander verbunden. In

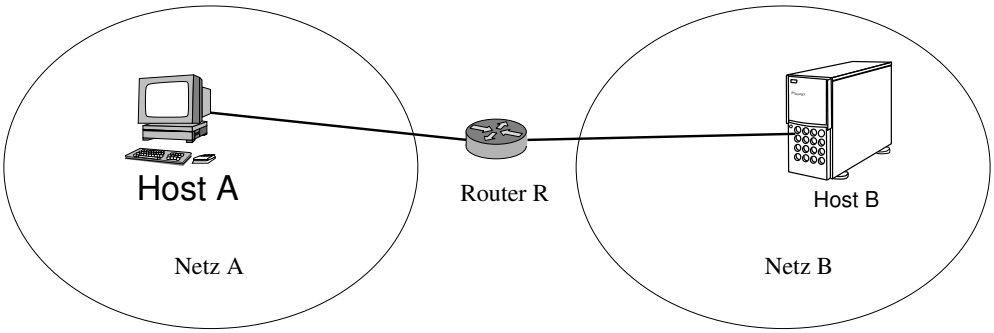

Abbildung 2.2: Kommunikation zwischen zwei Rechnern in unterschiedlichen Netzen

diesen Netzen befinden sich die Rechner *Host A* und *Host B*. Auf beiden Rechnern laufen Applikationen ab, die miteinander kommunizieren, z.B. in einer Client-Server Kommunikation.
Architekturen von Kommunikationssystemen sind in hierarchische Netzschichten untergliedert. Jede dieser Schichten stellt eine bestimmte Funktionalität zur Verfügung, die von der jeweils höher liegenden Schicht benutzt wird. Durch diese Hierarchie ist es möglich die u.U. große Komplexität von Kommunikationssystemen auf den jeweiligen Schichten zu abstrahieren.
Als bekanntestes Modell für Netzschichten ist hier das *ISO Reference Model of Open System Interconnection* [Com00] anzuführen. In diesem Modell werden Kommunikationssysteme durch sieben Netzschichten dargestellt.
Im Rahmen der Entwicklung des Internets kristallisierte sich aber ein gegenüber dem ISO-Modell leicht vereinfachtes Schichtenmodell heraus. Vor allem geprägt durch die Protokolle der TCP/IP-Familie wird heute, im Zusammenhang mit dem Internet, das TCP/IP 5-Schichten Referenzmodell verwendet [Com00].

In Abbildung 2.3 ist das Beispielszenario mit den Schichten des TCP/IP-Referenzmodells dargestellt*.
Die *Network Interface-Schicht* kümmert sich um die Übertragung von Datenpaketen in den jeweiligen Netzwerken. Ziel ist hier die Übertragung von Datenpaketen zwischen zwei (oder mehreren) Netzwerkkarten, die über ein Medium direkt miteinander verbunden sind. Als Beispiel führen wir Ethernet an.

*Zur Vereinfachung sind nur die oberen vier Software-basierten Schichten des TCP/IP-Referenzmodells dargestellt. Die Hardwareschicht fehlt in der Darstellung.

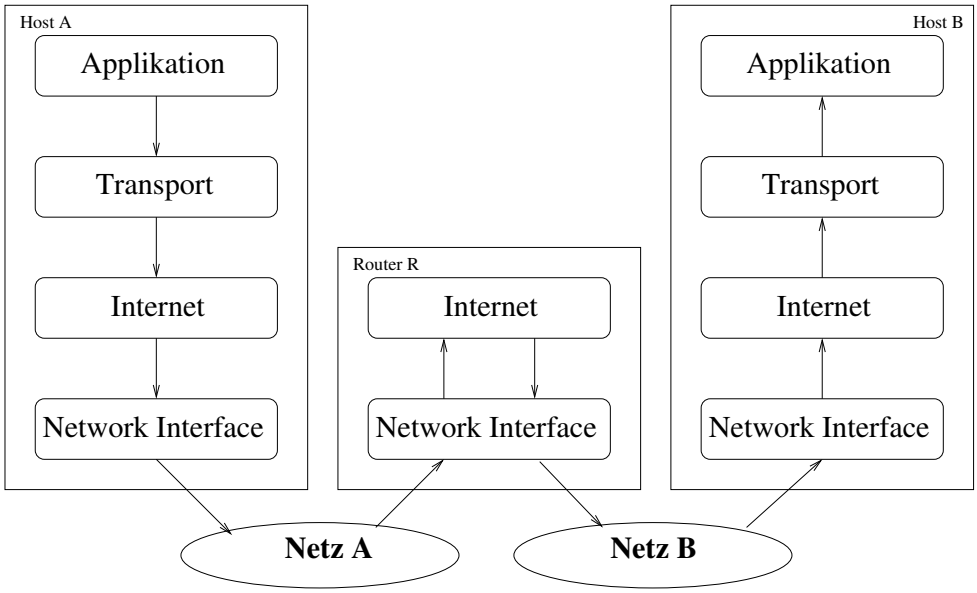

Abbildung 2.3: Die Schichtenarchitektur im Falle von Routing

Die Hauptaufgabe der *Internetschicht** ist es, Datenpakete zwischen Rechnern im Internet zu vermitteln. Basierend auf Routing-Algorithmen werden Datenpakete auf einer per Hop-Basis durch das Netz geroutet. In unserem Beispiel sorgt die Internetschicht dafür, dass Datenpakete von Host A nach Host B gelangen können. Im Unterschied zu anderen Kommunikationssystemen (z.B. X.25) ist der Datentransport auf der Vermittlungsschicht im Internet jedoch nicht zuverlässig. Aufgabe der Internetschicht ist, lediglich einen möglichen Weg für Datenpakete zur Verfügung zu stellen. Ob Datenpakete auf diesem Weg verloren gehen, wird nicht berücksichtigt.
In der Internet- bzw. Vermittlungsschicht wird nicht zwischen unterschiedlichen Applikationen unterschieden. Es können auf einem Computer mehrere Applikationen mit Kommunikationsverbindungen zu unterschiedlichen Rechnern parallel ablaufen. Die Unterscheidung und Trennung der u.U. zahlreichen Kommunikationsverbindungen, die von einem Computer ausgehen, ist Aufgabe der *Transportschicht*. Abhängig vom verwendeten Protokoll in der Transportschicht können in dieser Schicht zusätzlich auch Fehlertoleranzmechanismen eingebaut sein. Für den Fall des TCP-Protokolls gewährleistet die Transportschicht den zuverlässigen Transport von Datenpaketen, außerdem wird die korrekte Reihenfolge der Pakete sichergestellt.
In der *Applikationsschicht* schließlich laufen sog. *Userspace-Programme* (z.B. die Browsersoftware Netscape). Diese nutzen den von der Transportschicht zur Verfügung gestellten Dienst. Abhängig vom benutzen Protokoll in der Transportschicht muss in der Applikationsschicht der zuverlässige Datentransport implementiert werden. Dies ist z.B. für das UDP-Protokoll der Fall.
In Abbildung 2.3 sehen wir anschaulich das im Internet verwendete Prinzip der Ende-zu-Ende-Zuverlässigkeit. In der Regel wird der zuverlässige Datentransport ausschließlich durch die kommunizierenden Computer am Rande des Netzes sichergestellt. Dies geschieht entweder in der Transport- oder der Applikationsschicht der Kommunikationspartner. Die

*bzw. Vermittlungsschicht

Netze, durch die ausgetauschte Datenpakete der Kommunikationspartner geroutet werden, kümmern sich nicht darum, ob diese Pakete auch zuverlässig ankommen.
Durch diesen Mechanismus ist es möglich, IP-basierte Netze im Vergleich zu anderen Architekturen relativ günstig herzustellen und zu betreiben. Die (teure) Intelligenz liegt nicht im Netz, sondern in den Endsystemen. Dieser Punkt ist sicherlich einer der Erfolgsfaktoren für den Siegeszug der IP-Netze bzw. des Internets in den letzten Jahren.
Die Einfachheit der IP-Netze hat offensichtlich Vorteile, aber auch gewisse Nachteile. Hier sind Probleme des Quality of Service (QoS) anzuführen, aber auch die durch diese Arbeit verbesserten rudimentären Fehlertoleranzmechanismen des Internets.

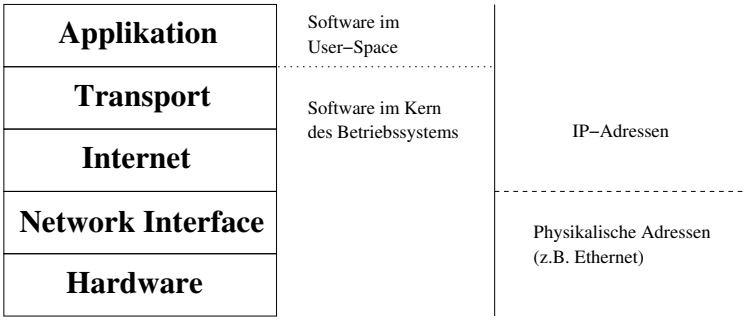

Abbildung 2.4: Einbettung der Netzschichten in ein Betriebssystem

In Abbildung 2.4 ist die Einbettung des TCP/IP-Referenzmodells in ein Betriebssystem im Allgemeinen dargestellt. Von den vier Softwareschichten des TCP/IP-Modells sind drei Schichten in den Kern des Betriebssystems integriert. Sowohl die Gerätetreiber in der Network Interface-Schicht, die Funktionalitäten des Internet-Protokolls in der Internetschicht, als auch die Mechanismen der Transportschicht sind ausschließlich Teile des Betriebssystems. Die Applikationsschicht ist über Schnittstellen, z.B. *Socket*-Funktionen, mit der Transportschicht verknüpft.
Die in Abschnitt 2.3 eingeführten Programmierbaren Netze bieten Dienste an, die in alle Softwareschichten des TCP/IP-Referenzmodells eingreifen können. Die vorgestellten, zugrundeliegenden programmierbaren Plattformen greifen somit in den Betriebssystemkern als auch im Userspace in den Kommunikationsfluss ein.

2.2 Overlay-Netze

Von Beginn an ist das Internet eng mit der Entwicklung von Overlay-Netzen verbunden. So war das Internet bzw. ARPANET anfangs selbst ein Overlay-Netz. Über das Telefonnetz wurden Rechner verschiedener Universitäten und Forschungseinrichtungen in den USA miteinander verbunden. Somit war das Internet ursprünglich ein Overlay-Netz über dem Telefonnetz. Im Laufe der Zeit wurden die Telefonleitungen durch explizite Datenleitungen ersetzt. Telefonleitungen werden heute, die Datenkommunikation betreffend, im Wesentlichen nur noch dazu benutzt, um Endsysteme in privaten Haushalten an das Internet anzuschließen. Aber auch auf diesem Sektor werden herkömmliche Telefonleitungen allmählich von neuen Technologien, z.B. DSL, verdrängt.

Wesentliche Meilensteine bei der Entwicklung des Internets wurden durch Overlay-Netze gesetzt. So ist z.B. das Domain Name System (DNS) ein Overlay-Netz. Auch der Austausch von E-Mails basiert im Internet auf einem Overlay-Netz.

Der in Abschnitt 2.1 diskutierte relativ einfache Aufbau von IP-basierten Netzen in Verbindung mit dem Konzept, die Intelligenz möglichst am Rande der Netze unterzubringen, ist sicherlich einer der Erfolgsfaktoren für die große Verbreitung und Akzeptanz des Internets. Der einfache Aufbau des Netzes führte aber schon bald an Grenzen, die möglichen Dienste betreffend. Ursprünglich war es z.B. nicht vorgesehen, im Internet Gruppenkommunikationsdienste zu unterstützen. Durch die zunehmende Akzeptanz und Verbreitung kam die Idee auf, das Medium Internet für diese neue Kategorie von Diensten zu nutzen.
Durch den Siegeszug des World Wide Web in den letzten 10 Jahren wurde das Internet immer stärker in der Industrie eingesetzt. Vor allem große, global operierende Unternehmen können vom Internet profitieren, indem sie (weltweit) verteilte Standorte ihres Unternehmens kostengünstig über das Internet vernetzen.
Diese zwei Phänomene, zum einen die gewünschte Einführung neuer Dienste, zum anderen die aufkommende Nutzung des Internets zur Vernetzung von Unternehmen, führten zu einer Vielzahl von auf die ursprüngliche Topologie des Internets aufgesetzten virtuellen Netzen bzw. *Overlay-Netzen*.
In diesem Abschnitt werden kurz die wesentlichen Kategorien von Overlay-Netzen im Internet diskutiert.
Zunächst präsentieren wir die ursprünglich als *Zwischenlösung* gedachten Overlay-Netze. Diese Kategorie von Overlay-Netzen wurde in der Forschung eingesetzt, um neue Dienste im Internet zu entwickeln und zu testen. Ziel der ersten in Betrieb gegangenen Overlay-Netze ist und war es, die entwickelten Dienste schließlich in die eigentliche Infrastruktur des Internets zu migrieren. Overlay-Netze sind ein möglicher Ansatz, diese Migrationen schrittweise durchzuführen, bis schließlich die jeweiligen Overlay-Netze nicht mehr nötig sind, da die gewünschte Funktionalität im Netz selbst zur Verfügung steht. Aus verschiedenen Gründen kristallisierten sich aber einige dieser Ansätze als ständige Provisorien heraus. Die grundlegenden technischen Probleme, neue Dienste betreffend, wurden gelöst. Aus nicht-technischen Gründen fand aber keine Migration der Technologien in die Infrastruktur des Internets statt.
Die zweite Kategorie von Overlay-Netzen ist unter dem Schlagwort *Virtuelle Private Netze* (VPN) bekannt. Hier ist in der Regel die kostengünstige Vernetzung von Teilen eines Unternehmensnetzes über das Internet gemeint. Die eigentliche Topologie bleibt dem Benutzer dabei verborgen.
Die dritte vorgestellte Kategorie fassen wir unter dem Begriff *Neuere Overlay-Netze* zusammen. Einer der Gründe für die Entstehung dieser Overlay-Netze liegt in der weiter oben angeführten Tatsache begründet, dass die Migrierung der als Zwischenlösung gedachten Overlay-Netze stagnierte. Ziel der neueren Overlay-Netze ist es, die gewünschte Funktionalität über zusätzliche Intelligenz am Netzrand zur Verfügung zu stellen. Vertreter dieser Art von Overlay-Netzen sind z.B. Ansätze des Application Layer Multicast. Auch die in den letzten Jahren sehr populär gewordenen Peer-to-Peer-Netze gehören dazu.

Mit der Einführung der als Zwischenlösung gedachten Overlay-Netze sieht man deutlich das Ziel, mehr Intelligenz im Netz zu verankern. In den letzten Jahren kam es aber teilweise zu einer Stagnation. Die Vorschläge wurden nicht umgesetzt. Eine Migration fand praktisch nicht statt (z.B. Multicast Backbone Overlay-Netz). In erster Linie zeichnen wir ökonomische

Gründe dafür verantwortlich.
Die neueren Overlay-Netze stellen eine Kehrtwendung in der Entwicklung dar, und sehen die für neue Dienste notwendige Intelligenz wieder am Netzrand in den Endsystemen. Die steigende Beliebtheit in den letzten Jahren von Ansätzen dieser Art scheint der Entwicklung Recht zu geben.

Die vorliegende Arbeit beschreibt einen Netzdienst, den kommunizierende Applikationen im Internet benutzen können, um die Wahrscheinlichkeit von Fehlern während der Kommunikation zu verringern. Der vorgeschlagene Dienst basiert auf kooperierenden Overlay-Netzen. Im Vergleich zu den skizzierten Ansätzen beschreiben wir einen Mittelweg. Grundsätzlich sehen wir die notwendige Intelligenz für unsere Architektur eher in den Randbereichen des Netzes. Wir schlagen jedoch vor, die notwendige Intelligenz in erster Linie in Access-Router bzw. Gateways zu verlagern. Endsysteme beziehen wir nur in geringem Maß ein.

2.2.1 IP-basierte Overlay-Netze als Zwischenlösung

Die Idee, neue Dienste durch Overlay-Netze anzubieten, führte zu einer Vielzahl von entwickelten Ansätzen. Das 1994 in Dienst genommene Multicast Backbone (M-Bone) [Eri94], stellt ein Overlay-Netz mit Multicast-Funktionalität zur Verfügung, basierend auf dem Distance Vektor Multicast-Protokoll (DVMRP) [WPD88]. Im M-Bone werden über das Internet verteilte Netze durch einen Tunnelmechanismus (IP-in-IP) [Sim95] miteinander verbunden. Innerhalb der einzelnen Netze ist Multicast-Funktionalität gegeben. Durch den Tunnelmechanismus können IP-Multicast Datenpakete dann von einem multicast-fähigen Netz zum anderen gelangen, obwohl zwischen den involvierten Netzen (in der Vermittlungsschicht) keine Multicast-Funktionalität vorhanden ist.
Ähnlich zum M-Bone ist das 1996 in Betrieb gegangene 6-Bone IPv6-Overlay-Netz [Fin03] aufgebaut. Dieses Overlay-Netz verbindet einzelne Netze, die auf Version 6 des Internet-Protokolls (IPv6) [DH98] basieren. Über Tunnelmechanismen sind diese IPv6-Netze, basierend auf dem gegenwärtigen Standard im Internet (IPv4), zu einem Overlay-Netz verbunden. Ähnlich zum M-Bone ist es somit möglich, verteilte "Inseln" im Netz, die einen bestimmten Dienst schon unterstützen, netzübergreifend zu einem virtuellen Netz zu verbinden. Die eigentliche Netztopologie und Funktionalität wird dabei durch Tunnelmechanismen abstrahiert.
Die Konfiguration der vorgestellten Ansätze ist komplex und daher fehleranfällig. Dies gilt im Besonderen für den dynamischen Fall, wenn einzelne Subnetze mit Overlay-Netzen verbunden bzw. von ihnen getrennt werden. Der im X-Bone-Projekt [Tou01] entwickelte Konfigurator vereinfacht die Konfiguration und das Management von Overlay-Netzen.

Die vorgestellten Overlay-Netze eignen sich zum einen gut für die Entwicklung und auch für den Betrieb von neuen Netzdiensten. Da eine Kapselung stattfindet, beeinflussen die Overlay-Netze die Funktionalitäten im Netz nicht. Andere Benutzer und Applikationen werden durch die beschriebenen Overlay-Netze nicht beeinträchtigt.
Zum anderen bieten die Overlay-Netze eine Möglichkeit, Netze schrittweise auf einen neuen Standard zu migrieren. So wäre es z.B. möglich, nach und nach in allen Netzen des Internets Multicast-Funktionen zu integrieren und diese an ein Overlay-Netz anzuschließen. Im Laufe der Zeit würde das Overlay-Netz allmählich verschwinden, bis schließlich alle Netze über

Multicast-Funktionalität verfügen. Der laufende Betrieb würde dadurch nicht beeinträchtigt. Somit eignen sich Overlay-Netze auch als Zwischenlösung bei der Einführung neuer Dienste.

2.2.2 IP-basierte Virtuelle Private Netze

Die ersten Ansätze Virtueller Privater Netze entstanden Anfang der 90er Jahre. Ein Beispiel ist die LAN-Emulation (LANE) über ATM [For95], die die Möglichkeit bietet, verteilte private Subnetze zu einem virtuellen Netz zu verbinden. Voraussetzung in diesem Fall ist, dass alle beteiligten privaten Subnetze an den ATM Backbone angeschlossen sind. Man spricht hier von Nativen Virtuellen Privaten Netzen.
IP-basierte VPN [GLH+00] ähneln der LANE. Sie setzen den Zugang aller Teilnehmer des VPN zum gleichen IP-basierten Netz bzw. dem Internet voraus.
Der Grundgedanke von IP-basierten Virtuellen Privaten Netzen ist, ein privates *Wide Area Network* (WAN) zu emulieren.

Die Struktur des gewünschten privaten WAN wird dabei über ein öffentliches WAN (z.B. das Internet) abgebildet. Hierzu werden virtuelle Verbindungen zwischen den einzelnen beteiligten privaten Subnetzen durch Tunnelmechanismen (z.B. IP-in-IP [Sim95]) eingesetzt. Die physikalische Topologie des öffentlichen WAN bleibt dem Benutzer dabei verborgen. Er sieht nur die Topologie des virtuellen privaten WAN.
VPN gehören ebenfalls in die Familie der Overlay-Netze. Wir sehen in ihrem Aufbau große Ähnlichkeiten zu den in Abschnitt 2.2.1 beschriebenen Overlay-Netzen. In einem VPN werden, ähnlich wie z.B. beim M-Bone [Eri94], verteilte Subnetze über Tunnelmechanismen zu einem virtuellen Netz verbunden.
Die Overlay-Netze verfolgen im Vergleich zu den VPN aber ein anderes Ziel: Overlay-Netze dienen in erster Linie zur Entwicklung oder Bereitstellung neuer Dienste. Die VPN hingegen wurden hauptsächlich aus folgenden Gründen eingeführt:

- Kosteneffiziente Verbindung von verteilten (Unternehmens-) Netzen
- Sichere Anbindung von Berechtigten an interne Firmennetze

Basierend auf diesem kommerziellen Hintergrund ergeben sich einige Anforderungen an VPN. Das Hauptaugenmerk liegt bei VPN-Architekturen auf der Sicherheit. Sowohl die Datensicherheit als auch die Robustheit gegen Denial of Service Attacken [KMR02] sind von zentraler Bedeutung für den Einsatz von VPN in einem professionellen Umfeld.
Die Vorteile von VPN liegen in erster Linie in der vereinfachten Administration des Gesamtnetzes [Zim99] als auch in der relativ einfachen Erweiterbarkeit.

2.2.3 Dienste durch Application-Layer Overlay-Netze

In den Abschnitten 2.2.1 und 2.2.2 werden Overlay-Netze vorgestellt, die verteilte, vollständige Subnetze mit bestimmten Eigenschaften zu virtuellen Overlay-Netzen verbinden. Dies wird durch Tunnelmechanismen gewährleistet, die im Allgemeinen Zugangspunkte zu den entsprechenden Subnetzen miteinander verbinden. An den Endsystemen findet in den

beschriebenen Ansätzen keine Modifikation statt; für sie herrscht Transparenz.
In diesem Abschnitt werden IP-basierte Overlay-Netze vorgestellt, die auf Endsystemen aufsetzen. Zuerst skizzieren wir hier kurz das Domain Name System (DNS) sowie die notwendige Architektur, um E-Mails im Internet zu verschicken.

Das DNS entstand um 1980. Basierend auf frühen Entwicklungsstufen (z.B. RFC 819 [SP82]) wurde es im Lauf der Jahre erweitert [Moc87a, Moc87b]. Grundgedanke des DNS ist, die für Menschen im täglichen Gebrauch umständlich zu benutzenden IP-Adressen in Namen aufzulösen. Das DNS identifiziert Rechner im Netz eindeutig durch Namen. So ist es mittels DNS z.B. möglich, den Webserver der TU München anstatt durch seine IP-Adresse (129.187.39.10) mittels des Namens (www.tum.de) weltweit eindeutig zu identifizieren.
Realisiert ist das DNS durch ein Overlay-Netz, das ausschließlich auf Endsystemen aufsetzt: den Name-Servern. Diese sind hierarchisch geordnet (von der Top Level-Domäne bis zu den Name-Servern der niedrigsten Stufe, die einzelne Rechner auflösen), und mittels des DNS-Protokolls verbunden.

Etwa zur gleichen Zeit wie das DNS entstand auch das heute verwendete Overlay-Netz zum Austausch von E-Mails im Internet. Auch in diesem Fall handelt es sich um ein Overlay-Netz, das auf Endsystemen aufsetzt: den Mail-Servern. Auf den Mail-Servern sind Fächer für den Posteingang und Postausgang der Teilnehmer untergebracht. Die Mail-Server organisieren die Weiterleitung von abgesandten E-Mails zu den Posteingängen der Empfänger. Basis für dieses Overlay-Netz ist das Simple Mail Transfer Protocol (SMTP) [Kle01].
Die zwei präsentierten Overlay-Netze sind klare Erfolgsfaktoren für die Entwicklung des Internets und ein Beweis für die Mächtigkeit von Overlay-Netzen.

In jüngster Zeit entsteht eine Vielzahl an Overlay-Netzen, die – ebenfalls wie das DNS – auf Endsystemen aufbauen. Die Gründe für den erfolgreichen Start dieses neuen Forschungsgebietes liegen zum einen in der Stagnation der Umsetzung der in Abschnitt 2.2.1 vorgestellten Ansätze. Zum anderen sind die als Zwischenlösung eingeführten Overlay-Netze relativ unflexibel. Im vorgestellten M-Bone Overlay-Netz wäre es z.B. relativ aufwendig, ein anderes Routing-Protokoll als das Gegenwärtige zu benutzen. Alle beteiligten Router müssten umkonfiguriert werden. Außerdem besteht für einen Benutzer keine Möglichkeit die angebotenen Dienste zu nutzen, wenn er in einem Subnetz konnektiert ist, das keinen Zugang zum M-Bone hat.
Aus diesen Gründen sind *Application Layer*-basierte Overlay-Netze entstanden. Für den Spezialfall der Multicast-Unterstützung sind als Vertreter hier z.B. die Architekturen Overcast [JGJ+00], ALMI [PSVW01] und End System Multicast [hCRSZ01] zu nennen.
Overcast ist ein Ansatz für Single Source Multicast-Anwendungen, d.h. ein Sender überträgt Daten an mehrere Empfänger. Der Fokus dieses Overlay-Netzes liegt in erster Linie im zuverlässigen Datentransfer (reliable Multicast) zwischen Sender und Empfänger. Die Skalierbarkeit des Overcast-Ansatzes wurde bis in die Größenordnung von einigen hundert beteiligten Knoten experimentell validiert.
Mit der *Application Level Multicast Infrastructure* (ALMI) schlagen Pendarakis et al. einen Ansatz zur Gruppenkommunikation vor. Der Fokus dieser Arbeit liegt auf der effizienten Konstruktion des Multicast-Baumes. Die unterstützte Gruppengröße für die beteiligten

Multicast-Gruppen liegt in der Größenordnung von einigen zehn Teilnehmern.
Der *End System Multicast* Ansatz von Yang-hua Chu et al. schlägt ebenfalls ein Application Layer Multicast Overlay-Netz für Multi Source-Multicast-Anwendungen vor. Insbesondere sollen durch die Architektur Videokonferenzen unterstützt werden. In Erweiterung zu ALMI wird in diesem Ansatz die Verzögerung und mögliche Übertragungsrate in den dynamischen Aufbau des Overlay-Netzes mit einbezogen. Die Skalierbarkeit steht hier nicht im Vordergrund, da die Zahl der möglichen Teilnehmer durch die Anwendung begrenzt ist.
Diese beispielhaft skizzierten Architekturen zeigen den Vorteil der Application Layer Overlay-Netze auf. Abhängig von der konkreten Anforderung der Anwendung (z.B. Multi Source oder Single Source) kann ein maßgeschneiderter Multicast-Dienst angeboten werden. Diese Flexibilität wäre im Allgemeinen nur mit großem Aufwand zu realisieren.
Natürlich gibt es auch gewisse Nachteile [CW02] gegenüber den reinen IP-Multicast-Architekturen. Abhängig vom konkreten Szenario können diese aber u.U. vernachlässigt werden.

Der Vollständigkeit halber erwähnen wir in diesem Zusammenhang auch die in letzter Zeit aufgekommene Technologie des Grid-Computing [FK99]. Die Idee von Grid-Computing ist, aus vielen über das Internet verteilten Computern einen *virtuellen Supercomputer* zu bilden. Dem Benutzer bleibt dabei die physikalische Unterteilung in viele Einzelmaschinen verborgen; er sieht nur einen leistungsstarken (virtuellen) Rechner.
Der Vorteil von Grid-Computing liegt in der Möglichkeit, die Nutzung von Rechen- und Speicherressourcen zu optimieren. So gibt es bereits erste kommerzielle Ansätze [ibm03], die es ermöglichen, im Netz vorhandene Ressourcen für die Abarbeitung von Rechenprozessen zu nutzen.
Gegenwärtiger Stand der Forschung ist, Grid-Computing Systeme auf offenen Standards und Protokollen aufzubauen, wie z.B. die Open Grid Service-Architektur (OGSA) [ogs03], um virtuelle Rechensysteme im heterogenen und verteilten Umfeld zu ermöglichen.

2.2.4 Peer-to-Peer-Netze

Peer-to-Peer (P2P) Netze fallen ebenfalls in die in Abschnitt 2.2.3 präsentierte Kategorie der Application Layer Overlay-Netze. Wegen der herausragenden Bedeutung der P2P-Netze für diese Arbeit gehen wir in diesem Abschnitt gesondert auf P2P-Netze ein.

Peer-to-Peer Netze bestehen aus Teilnehmern (*Peers*), die in einem Application-Layer Overlay-Netz direkt oder indirekt miteinander verbunden sind. Diese Teilnehmer stellen den jeweiligen anderen Peers in ihrem P2P-Netz festgelegte lokale Ressourcen zur Verfügung. Dies kann z.B. Information bzw. Content sein. In diesem Fall spricht man von File Sharing P2P-Netzen. Grundprinzip dieser Art von P2P-Netzen ist der Community-Gedanke. Jeder der Peers stellt seine lokalen Ressourcen den anderen Peers zur Verfügung. Dafür erlangt er im Ausgleich ebenfalls Zugang zu deren Ressourcen.
Gemäß Schollmeier [Sch01] können P2P-Netze grundsätzlich in zwei Arten unterschieden werden:

- Hybride P2P-Netze

- Pure P2P-Netze

Als prominentesten Vertreter der hybriden P2P-Netze führen wir hier Napster [nap01] an. Schollmeier definiert hybride P2P-Netze dadurch, dass mindestens eine zentrale Instanz im P2P-Netz vorhanden sein muss, ohne die zumindest Teile eines gewünschten P2P-Dienstes nicht zur Verfügung gestellt werden können. Im Falle von Napster ist dies z.B. eine zentrale Datenbank, die die Information zur Verfügung stellt, auf welchen Teilnehmern des Napster P2P-Netzes der gewünschte Content zur Verfügung steht. Ein Client, der einen bestimmten Content im P2P-Netz sucht, muss somit zuerst eine Anfrage an die zentrale Datenbank stellen. Von der Datenbank wird er dann auf die entsprechenden Teilnehmer im P2P-Netz verwiesen, die den gewünschten Content zur Verfügung stellen. Ohne diese zentrale Instanz ist somit der gewünschte P2P-Dienst bei Napster nicht verfügbar.
Ein Vertreter purer P2P-Netze ist z.B. das Gnutella-Netz [Kan01]. Pure P2P-Netze zeichnen sich dadurch aus, dass alle teilnehmenden Peers gleichberechtigt sind. Eine zentrale Instanz fehlt. Die notwendigen Management- und Routing-Funktionen müssen von den teilnehmenden Peers in verteilter Weise selbst zur Verfügung gestellt werden.

Pure P2P-Netze können weiter unterteilt werden in:

- Unstrukturierte P2P-Netze
- Strukturierte P2P-Netze

Gnutella ist ein Vertreter der unstrukturierten puren P2P-Netze. Jeder der teilnehmenden Peers im Netz ist gleichberechtigt. Die physikalische Topologie der einzelnen Peers im Netz spielt beim Aufbau des P2P-Netzes keine Rolle. Auch die von den jeweiligen Peers zur Verfügung gestellte Information wird ebenfalls nicht in den Aufbau des P2P-Netzes mit einbezogen. Gnutella baut das P2P-Netz zwischen den einzelnen Peers zufällig auf. Dieser unstrukturierte Aufbau kann z.T. zu einer sehr ineffizienten Topologie des P2P Overlay-Netzes führen [SH03]. Dies geht u.U. sogar so weit, dass sich Routingschleifen bei der Suche im Overlay-Netz bilden [SS02].
Die strukturierten P2P-Netze setzen an diesem Punkt an. Vorrangiges Ziel ist, das Suchen bzw. die Verteilung von Content im P2P-Netz effizienter zu gestalten. Zwei unterschiedliche Arten, P2P-Netze zu strukturieren, werden vorgeschlagen:

- Ressourcenbasierte Strukturierung
- Topologiebasierte Strukturierung

In der ressourcenbasierten Strukturierung wird eine Relation zwischen der Art der Ressource und der Lage der Ressource eingeführt. Ein Vertreter dieser Kategorie ist die Chord P2P Architektur [SMK+01]. Mittels Hash-Funktionen wird definierter Content auf die Adresse bestimmter Peers abgebildet. Somit ist es möglich, bestimmten Content effizienter zu finden als in unstrukturierten P2P-Netzen.
Als Erweiterung von P2P-Netzen wird ein Ansatz topologiebasierter Strukturierung in [Fuh03] vorgeschlagen. Grundprinzip hier ist, bestimmten Content zuerst bei Peers in lokaler Nähe des anfragenden Peers zu suchen. Liegt der gewünschte Content auf einem Peer in lokaler Nähe tatsächlich vor, dann bringt dieses Verfahren ebenfalls Effizienzvorteile. Im unstrukturierten Fall würde der Content u.U. von einem topologisch sehr weit

entfernten Peer angefordert werden. Ein topologisch näher gelegener Peer, der ebenfalls den gewünschten Content zur Verfügung stellt, würde u.U. nicht gefunden werden.

Das Prinzip der P2P-Netze wird in der vorliegenden Arbeit zur Signalisierung genutzt. Wir führen zu diesem Zweck hierarchisch strukturierte Overlay-Netze ein, die in den Zugangsbereichen des Internets auf Netzzwischensystemen alloziert sind.

2.3 Programmierbare und Aktive Netze

Mit der zunehmenden Popularität und Verbreitung des Internets entstand in den letzten Jahren der Wunsch, neben den Diensten der ersten Generation im Internet (z.B. E-Mail, FTP, WWW) auch Dienste der zweiten Generation (z.B. Audio- oder Video-Streaming, VoIP, Telekonferenzen) anzubieten. Darüber hinaus erfreut sich seit geraumer Zeit die drahtlose Anbindung an das Internet immer größerer Beliebtheit. Zu relativ geringen Kosten ist es heute möglich, sich z.B. mittels Wireless LAN drahtlos mit dem Internet zu verbinden.
Diese zwei Phänomene, die immer anspruchsvoller werdenden Dienste in Kombination mit der mobilen drahtlosen Anbindung, stellen die Architektur des Internets vor ein fundamentales Problem.
Wie in Abschnitt 2.1 skizziert, ist das Internet ein Zusammenschluss vieler unabhängiger Netze. In diesen Netzen sind Endsysteme (z.B. Arbeitsplatzrechner, Server) über Router miteinander verbunden. Wie in Abbildung 2.3 dargestellt, ist ein Großteil der notwendigen Intelligenz in den Endsystemen vorgesehen. Die Router sind ursprünglich nur für die Weiterleitung von Datenpaketen vorgesehen. Diese Weiterleitung von Daten ist unzuverlässig. Router leiten Datenpakete nach dem *Best Effort*-Prinzip weiter. Ist dies nicht möglich, da z.B. in einem Router die Warteschlangen in den Eingangs- bzw. Ausgangspuffern voll sind, werden neu ankommende Datenpakete im Netz von den Routern verworfen. In den Endsystemen müssen diese Datenausfälle festgestellt und entsprechend behandelt werden.
Für Dienste der ersten Generation war dieses Verhalten des Netzes noch kein allzu großes Problem. Verlorene Datenpakete können von den Endsystemen erneut angefordert bzw. gesendet werden und treffen somit statistisch bei den kommunizierenden Endsystemen ein.
Die Dienste der zweiten Generation jedoch sind auch durch gewisse Anforderungen an die Verzögerung von Datenpaketen gekennzeichnet. So ist es zum Beispiel bei einer Telefonkonferenz über das Internet nicht ohne weiteres möglich, ein verloren gegangenes Paket mit Sprachdaten erneut anzufordern. Der Gesprächsfluss würde unterbrochen, die Verzögerung auf Teilnehmerseite als störend empfunden.
Abgesehen von der Verzögerung von Datenpaketen gibt es noch weitere Anforderungen von Diensten an das Netz. Dies ist zum einen die effiziente Unterstützung von Gruppenkommunikation, sowohl für kleine Gruppen (z.B. Telekonferenz), als auch für mittlere und große Gruppen (Live-Fernsehen über das Internet). Zum anderen stellt auch die mobile Anbindung von Teilnehmern gewisse Anforderungen an das Netz [KP01].

Zusammenfassend kann man sagen, dass die Vielzahl an neuen Diensten es erfordert, mehr Intelligenz im Netz zu verankern. Die bisherige Lösung, diese Aufgabe in die Endgeräte zu verlagern, genügt den Anforderungen der neuen Dienste nicht.

Der Wunsch zusätzliche Funktionalität und somit Intelligenz im Netz zu verankern, ist also unstrittig. Die Überlegung, die sich in diesem Zusammenhang stellt, ist, wie dies praktisch umgesetzt werden kann.
Für neue Dienste greift der ursprüngliche Ansatz des Internets "One Size fits all" nicht mehr. Dies liegt in den unterschiedlichen Anforderungen der neuen Dienste begründet. So stellt z.B. der Dienst "Fernsehen über das Internet" andere Anforderungen an das Netz als ein VoIP-Dienst.
Gängige Praxis in Kommunikationssystemen ist, neue Anforderungen bzw. Netzdienste zuerst zu standardisieren. Ist die Standardisierung abgeschlossen, erfolgt die Migration der neuen Netzdienste. Am Beispiel der GSM-Netze zeigte sich, dass ein in der Breite akzeptierter Standard eine sehr gute Basis sein kann für eine spätere erfolgreiche Umsetzung.
Im Umfeld des Internets zeigt sich der Ansatz der Standardisierung aber bisher nicht sehr erfolgreich. Die Gründe dafür sind mannigfaltig. Einmal spielt die Dauer eines Standardisierungsprozesses eine Rolle, da sich Standardisierungen in der Regel über Jahre hinziehen. Dem gegenüber steht die quasi ständige Entwicklung von neuen Diensten, die u.U. neue Anforderungen an das Netz stellen. Des Weiteren sind im Internet ungleich mehr Parteien beteiligt als es im Vergleich bei z.B. GSM- oder PSTN-Netzen sind. Während die Anzahl der Service Provider und Hersteller im Bereich der herkömmlichen Telefonie (Mobil- oder Festnetz) weltweit relativ überschaubar ist, ist dies im Internet durch den heterogenen Aufbau nicht der Fall. Allein schon die große Anzahl an beteiligten Parteien, mit z.T. unterschiedlichen Interessen, macht eine in der Breite akzeptierte Einigung auf einen Standard schwierig.

Die Netzbetreiber bzw. die Anbieter von Diensten stehen somit vor einem Dilemma. Einerseits ist es nötig, zusätzliche Intelligenz im Internet bereitzustellen, um die Vielzahl an neuen Diensten besser unterstützen zu können, bzw. bestimmte Dienste überhaupt anbieten zu können. Wie an der lahmenden Umsetzung verschiedener Dienste abzulesen ist, ist es andererseits praktisch sehr schwierig, dies in der Breite tatsächlich durchzuführen.
Eine Lösung dieses Dilemmas könnte in den Programmierbaren und Aktiven Netzen liegen [Har02].

2.3.1 Motivation und Eigenschaften

Programmierbare und Aktive Netze sind ein relativ junges Forschungsgebiet. Die ersten Arbeiten wurden erst vor ein paar Jahren vorgestellt. Die fundamentalen Beiträge von Wetherall, Campbell und Alexander [WGT98, WLG98, CMK+99, AAH+98] gegen Mitte des letzten Jahrzehntes wiesen den Weg für zahlreiche weitere Arbeiten auf diesem Gebiet.
Programmierbare und Aktive Netze sind gekennzeichnet durch folgende Eigenschaften:

- Sie bieten die Möglichkeit, Intelligenz im Netz zu platzieren.
- Netze können an die jeweiligen Anforderungen von Applikationen dynamisch angepasst werden.
- Neue Netzdienste können relativ schnell, auch unter Umgehung von Standardisierungen, angeboten werden $\Longrightarrow$ Rapid Service Creation.

Durch Programmierbare und Aktive Netze kann Intelligenz im Netz platziert werden durch die Möglichkeit, einen Programmcode auf bestimmten Knoten im Netz auszuführen. Dieser Programmcode repräsentiert zusätzliche Netzdienste zum reinen Transport von Daten.
Abhängig von der jeweiligen Architektur kann ein Außenstehender (z.B. der Benutzer am Rand des Netzes oder ein Administrator) das Netz dann mittels Wahl des auszuführenden Programmcodes auf seine speziellen Bedürfnisse anpassen. Diese Anpassung des Netzes an Anforderungen kann flexibel und dynamisch erfolgen.
Die eigentliche Implementierung des Netzdienstes kann genau auf die entsprechenden Anforderungen zugeschnitten werden. Mit Programmierbaren und Aktiven Netzen ist eine nahezu unbegrenzte Vielfalt an Netzdiensten möglich. Da für jede Anforderung ein dedizierter Netzdienst entworfen werden kann, ist das Problem der langen Dauer der Standardisierung von zusätzlichen Netzdiensten weitgehend umgangen. Somit eignet sich der Ansatz Programmierbarer und Aktiver Netze auch dazu, relativ schnell und flexibel neue Netzdienste anzubieten. Es sollten lediglich noch durch ein Rahmenwerk die Eigenschaften der Basisfunktionalität festgelegt werden, um z.B. zu verhindern, dass sich Netzdienste gegenseitig behindern. Auch die Sicherheit bzw. die möglichen Bedrohungen durch diese Art von Netzdiensten sollten darin behandelt werden.

Die wesentlichen Elemente der Programmierbaren und Aktiven Netze sind die programmierbaren und aktiven Netzknoten und die Netzdienste, die auf diesen Knoten dynamisch bereitgestellt werden können.
Diese Elemente werden im Folgenden erläutert.

2.3.2 Funktionalität

Die grundlegende Eigenschaft von Programmierbaren und Aktiven Netzen ist die Möglichkeit, einen dedizierten Programmcode an wählbaren Orten im Netz auszuführen.
Prinzipiell kann dies auf zwei Arten erfolgen:

- Explizite Signalisierung und Ausführung.
- Implizite Signalisierung und Ausführung.

Hier unterscheiden sich *Programmierbare Netze* von *Aktiven Netzen*.
In Programmierbaren Netzen, bzw. bei der Ausführung programmierbarer Dienste findet zuerst eine explizite Signalisierung statt. Ein *Netzkonfigurator* (z.B. ein Administrator oder eine Applikation) will die Eigenschaften des Netzes gemäß seinen Anforderungen dynamisch anpassen. Er signalisiert diesen Wunsch explizit an das Netz. Nach dem positiven Bescheid von Sicherheits- und Verwaltungseinheiten wird ein Programmcode auf die gewünschten *programmierbaren Netzelemente* geladen und ausgeführt. Dieser u.U. im Netz verteilt ausgeführte Programmcode repräsentiert einen Netzwerkdienst mit den gewünschten Eigenschaften.
Nach erfolgter Konfiguration des Netzes können Applikationen den neuen, nun zur Verfügung stehenden Netzwerkdienst nutzen.

Bei Aktiven Netzen findet keine explizite Signalisierung statt. Die Signalisierung und auch der gewünschte Programmcode sind in die Datenpakete, sog. *aktive Pakete*, der Applikation

mit eingebettet. Auf den einzelnen *aktiven Netzelementen* im Netz laufen standardisierte Prozesse ab, die aktive Pakete abfangen und verarbeiten. Der in den aktiven Paketen mitgelieferte Programmcode wird, ebenfalls nach positivem Bescheid von Sicherheits- und Verwaltungseinheiten, von diesen Prozessen auf den aktiven Netzelementen ausgeführt. So ist es möglich, Datenpaketen implizite Anweisungen mitzugeben, wie diese Pakete im Netz behandelt werden sollen. Dadurch können Datenpakete differenziert werden.

In der grundlegenden Eigenschaft, einen Programmcode im Netz auszuführen, unterscheiden sich Programmierbare und Aktive Netze nicht voneinander. Praktisch gesehen liegt aber in der Mächtigkeit möglicher Netzdienste ein großer Unterschied zwischen den beiden Architekturen. Offensichtlich ist die Größe des Programmcodes in aktiven Paketen sehr begrenzt. Die durch das Ethernet in den Randbereichen des Internets vorgegebene maximale Paketgröße beläuft sich auf 1500 Byte. Abzüglich Headern und Nutzdaten der Applikation bleibt somit nur wenig Raum für Signalisierung und Programmcode in aktiven Paketen.
Dies hat natürlich unmittelbare Auswirkung auf die Mächtigkeit möglicher Dienste. Mögliche Netzdienste in Aktiven Netzen sind z.B. Netz-Management, Multicast, einfache Quality of Service-Dienste. Die Programmierbaren Netze sind im Vergleich dazu ungleich mächtiger. Die Funktionalität der programmierbaren Netzdienste ist im Wesentlichen nur begrenzt durch die zur Verfügung stehenden Ressourcen auf den programmierbaren Netzelementen bzw. im Netz.
Kombinationen aus Programmierbaren und Aktiven Netzen sind ebenfalls denkbar. So ist es z.B. möglich, die beschriebenen Prozesse der aktiven Netzelemente in einer programmierbaren Umgebung zu laden. Aus diesem Grund, in Verbindung mit den identischen grundlegenden Eigenschaften, differenzieren wir in dieser Arbeit begrifflich nicht zwischen den Ansätzen Programmierbarer Netze und Aktiver Netze. Wir verwenden in der Nomenklatur den allgemeinen Ansatz der *Programmierbaren Netze*, die bei Bedarf auch aktive Eigenschaften haben können.

2.4 Netzdienste mit Programmierbaren Netzen

In Abschnitt 2.3 wurden grundsätzliche Konzepte der Programmierbaren Netze vorgestellt. Dieser Abschnitt gibt eine exemplarische Einführung in, durch Programmierbare Netze realisierte, mögliche Netzdienste.
Mit Hilfe des programmierbaren Ansatzes können Netze dynamisch an bestimmte Anforderungen, sowohl die Applikationen als auch den Betrieb des Netzes betreffend, angepasst werden. Konkret bedeutet das die verteilte Möglichkeit des Eingriffs in Datenströme in den folgenden in Abbildung 2.3 skizzierten Schichten:

- Applikationsschicht
- Transportschicht
- Internetschicht
- Network Interface-Schicht

Datenpakete können auf programmierbaren Knoten innerhalb des Netzes auf all diesen Schichten bearbeitet (z.B. verändert) werden.
Durch Programmierbare Netze ist zum einen die Möglichkeit gegeben, in die Vermittlung (Routing) bzw. den Transport von Datenpaketen einzugreifen. Darüber hinaus können programmierbare Knoten bzw. Dienste auch in die Applikationsschicht eingreifen.
Die Möglichkeit, in die Network Interface-Schicht einzugreifen, ist im Festnetz eher von untergeordneter Bedeutung, im drahtlosen Fall ergeben sich aber durchaus einige Anwendungen [SRBW01].
Ein besonderer Vorteil liegt sicher auch in möglichen Crosslayer-Diensten. So können programmierbare Dienste in einem Prozessschritt Informationen aus allen zugängigen Schichten gleichzeitig in die weitere Verarbeitung mit einbinden und auch auf allen zugängigen Schichten Veränderungen einfließen lassen.

Grundsätzlich können Netzdienste in Programmierbaren Netzen in zwei Kategorien unterschieden werden:

- Netzdienste, die allgemein den Betrieb des Netzes verbessern.
  - Netzmanagement.
  - Security: Angriffserkennung.
- Netzdienste, die direkt Applikationen unterstützen.
  - Neue Datentransport- bzw. Verteilmechanismen.
  - Unterstützung von contentspezifischen Anforderungen.

Mit Diensten in Programmierbaren Netzen kann man somit die Gegebenheiten des Netzes an Anforderungen von Applikationen explizit anpassen. Des Weiteren sind auch Netzdienste möglich, die den Betrieb von Netzen allgemein verbessern.
Auf diese Kategorien gehen wir im Folgenden ein.

2.4.1 Allgemeine Verbesserung des Netzbetriebs

Unter Netzdiensten, die den Betrieb des Netzes allgemein verbessern, fassen wir Dienste zusammen, die *nicht* in direkter Aktion mit Benutzer-Applikationen stehen.
So gibt es eine Reihe von Vorschlägen, die auf das Netzmanagement durch programmierbare bzw. aktive Dienste eingehen.
Raz und Shavitt [RS99] schlagen ein effizienteres Netzmanagement durch Programmierbare Netze vor. Die Architektur verteilt gegenwärtig zentralisierte Managementfunktionen im Netz. Der Vorteil liegt ähnlich wie bei agentenbasierten Systemen zum einen in kleineren Regelkreisen, die ein besseres zeitliches Verhalten gewährleisten. Zum anderen kann die beschriebene Architektur effizienter mit Ressourcen im Netz umgehen, da explizit Kosten (z.B. CPU-Last, Übertragungsrate) bei der Programmierung der Netzmanagement-Dienste berücksichtigt werden müssen. Die Autoren sehen dies als Vorteil gegenüber herkömmlichen, abstrahierenden Netzmanagement-Architekturen, die z.B. auf CORBA basieren.
Der Vorschlag von Moore et al. (z.B. [MMN02]) zielt insbesondere auf die Ressourcennutzung von Netzmanagementdiensten in aktiven Netzen. Die beschriebene Architektur stellt

ein System zur Verfügung, in dem die notwendigen Ressourcen schon vor der Ausführung der Managementdienste vorhergesagt werden können. Dies wird durch die eigens entwickelte Sprache "Safe and Nimble Active Packets" (SNAP) in den aktiven Datenpaketen gewährleistet. Der Vorteil dieses Systems liegt darin, dass bereits vor der Ausführung der Programme auf den aktiven Knoten klar ist, wie viele Ressourcen benötigt werden. Stehen diese nicht zur Verfügung, kann entsprechend reagiert werden. Somit ist gewährleistet, dass durch das aktive Netzmanagement der Betrieb des Netzes nicht zum Erliegen kommt.

Außerdem wurden in jüngster Zeit Programmierbare Netze auch vorgeschlagen, um Angriffe wie z.B. Denial of Service (DoS) Attacken auf Server zu erkennen und diese schon im Netz zu bekämpfen.
Die FIDRAN-Architektur [HJS03] ist ein auf Programmierbaren Netzen basierender Netzdienst, der im Netz Sicherheitsfunktionen zur Verfügung stellt. Der beschriebene Dienst eignet sich für die Erkennung und Bekämpfung von Angriffen im Netz, insbesondere auch für die Bekämpfung der kürzlich häufiger auftretenden Distributed Denial of Service (DDoS) Attacken. Die Dienstarchitektur sieht zum einen vor, FIDRAN-Software-Module auf Gateways zu installieren und somit konnektierte Sub-Netze zu schützen. Zum anderen ist es ebenfalls möglich, die Software auf speziell zu schützenden einzelnen Rechnern (z.B. Servern) zu platzieren. Das modulare Konzept von FIDRAN erlaubt zum einen große Flexibilität, da verschiedene existierende Sicherheitsmechanismen integriert werden können. Des Weiteren ist es durch die Modularität ebenfalls möglich während der Laufzeit das System anzupassen. Insbesondere bei vorher unbekannten Angriffsmustern kann der Netzdienst somit sehr schnell an neue Voraussetzungen angepasst werden.
Über ein Management- und Steuerungsmodul kooperieren (u.U. im Netz verteilte) FIDRAN-Komponenten bzw. können von einem Administrator konfiguriert werden. Die Funktionsweise des Dienstes kann grob in zwei Teile eingeteilt werden: die Angriffserkennung und die Behandlung der erkannten Angriffe. In beiden funktionellen Teilen ist die schnelle, verteilte dynamische Konfiguration des Netzdienstes von Vorteil. Wird ein neues Angriffsmuster bekannt, kann ein Administrator die FIDRAN-Angriffserkennung während des Betriebes rasch auf den neuen Stand bringen. Ist ein Angriff erkannt, ist eine schnelle Reaktion mit entsprechender Anpassung des Netzdienstes natürlich von Vorteil.
Karnouskos [Kar01] beschreibt einen grundsätzlich ähnlichen Ansatz. Fokus dieses Vorschlags ist allerdings die effiziente, agentenbasierte Signalisierung und Kontrolle des programmierbaren Netzdienstes. Die funktionellen Einheiten des Dienstes, die Erkennung der Angriffe und deren Behandlung werden durch kooperierende Software-Agenten gewährleistet. Dieser Ansatz verbessert die Skalierbarkeit. Darüber hinaus werden die relativ flexiblen Software-Agenten eingesetzt, um heterogene programmierbare Plattformen miteinander zu verbinden. Somit kann die Standardisierung von Schnittstellen umgangen werden.

Durch diese vorgestellten Netzdienste wird der Betrieb und das Verhalten des Netzes verbessert, ohne in direkte Aktion mit den Benutzerapplikationen zu treten. Die Applikationen können von diesen Diensten profitieren, indem z.B. eine DoS-Attacke verhindert wird und somit die Benutzer-Applikation ungehindert weiterlaufen kann.
Es gibt eine Vielzahl an weiteren Diensten, die den Netzbetrieb allgemein betreffen. So ist es möglich, neue Routing-Dienste [KRGP02] im Netz mittels programmierbarer Netze zur Verfügung zu stellen, die z.B. Quality of Service-Eigenschaften haben. Ein weiteres Beispiel

wäre die Unterstützung durch Programmierbare Netze im Mobilfunk [PW02].

2.4.2 Direkte netzseitige Unterstützung von Applikationen

Die zweite Kategorie von Netzdiensten unterstützt Applikationen direkt. Zum einen gibt es durch Programmierbare Netze die Möglichkeit, neue Datentransport- bzw. Verteilmechanismen im Netz zur Verfügung zu stellen, zum anderen kann das Netz auch direkt an contentspezifische Anforderungen von Applikationen angepasst werden. Auf die Vielzahl an möglichen Diensten gehen wir in diesem Abschnitt ebenfalls exemplarisch ein.
Backx et al. schlagen vor, auf programmierbaren Knoten im Netz Dokumente aus HTTP-Strömen zu cachen. Ziel der vorgeschlagenen Dienstarchitektur *Adaptive Distributive Caching* [BLP+02] ist zum einen die Minimierung der Übertragungsrate im Netz, zum anderen die Verkürzung der Übertragungszeit von Dokumenten bei Anfragen. Diese Ziele entsprechen den Zielen von regulären Cacheing-Systemen. Die Besonderheit der vorgeschlagenen Architektur ist, im Vergleich zu herkömmlichen Cacheing-Systemen, die Platzierung der programmierbaren Caches im Netz. Anfragen nach Dokumenten können somit im Netz durch den programmierbaren Dienst abgefangen und bearbeitet werden. Der vorgeschlagene Dienst sieht vor, dass eine Vielzahl von verteilten programmierbaren Knoten teilnimmt. Somit ergeben sich gegenüber herkömmlichen Cacheing-Systemen Skalierbarkeitsvorteile.
Der in [Fuh03] vorgeschlagene Dienst in Programmierbaren Netzen unterstützt ebenfalls direkt Applikationen. Fuhrmann schlägt vor, unstrukturierte, auf Endsystemen basierte P2P-Netze (siehe Abschnitt 2.2.4) durch einen Netzdienst zu strukturieren. Am Beispiel des Gnutella P2P-Netzes [gnu] wird gezeigt, wie der Aufbau des P2P-Netzes durch Netzdienste dahingehend manipuliert werden kann, dass sich bevorzugt Peers in physikalischer Nähe miteinander verbinden. Somit wird die physikalische Topologie des Netzes im P2P Overlay-Netz berücksichtigt. Der Autor führt an, dass die damit erreichbare Strukturierung des P2P-Netzes, Ressourcen im Netz sparen kann.
Die in [LPP+01] vorgeschlagene *Active Grid Architecture* schlägt eine netzseitige Unterstützung von Grid-Computing-Systemen (siehe Abschnitt 2.2.3) vor. Die Architektur schlägt ein Overlay-Netz vor, das auf programmierbaren Knoten basiert. Zwei wesentliche Dienste können Grid Computing Systemen durch dieses Overlay-Netz angeboten werden. Zum einen wird ein Netzdienst angeboten, der eine effiziente, auf verlässlichen Multicast basierende Signalisierung zur Verfügung stellt. Zum anderen wird ein Netzdienst bereitgestellt, der an Bedürfnisse von Grid-Applikationen angepasste Quality of Service (QoS) Eigenschaften hat.
Die von Harbaum et al. vorgeschlagene heterogene Gruppenkommunikation durch Netzdienste [HSWZ01] baut ebenfalls auf Programmierbaren Netzen auf. Basierend auf einer vorhandenen IP-Multicast-Infrastruktur werden im Netz verteilt QoS-Reduktionen eines ursprünglich hochqualitiven Multicast-Datenstroms vorgenommen. Diese QoS-Reduktionen passen den Datenstrom individuell an die Bedürfnisse von Empfängern an. Der Vorteil liegt insbesondere darin, dass zum einen Empfänger, die entsprechende Vorausetzungen haben, den hochqualitativen Datenstrom geliefert bekommen. Zum anderen können adaptierte Datenströme auch von Endsystemen empfangen werden, die nicht über die notwendigen Ressourcen verfügen, um den originalen Datenstrom zu empfangen und darzustellen. Ein Bei-

spiel hierfür wäre die dynamische Reduktion der Bitrate des Datenstroms, um es zusätzlich zu hochbitratig an das Internet angebundenen Empfängern auch mobilen Teilnehmern zu ermöglichen, den Datenstrom, wenn auch adaptiert, zu empfangen.
Applikationen aus der Kategorie der Gruppenkommunikation benötigen Mechanismen, um Daten an Teilnehmer effizient verteilen zu können. Diese Methoden können im Netz unabhängig von den Applikationen zur Verfügung gestellt werden, wie diverse Ansätze des IP-Multicast vorschlagen [Ram00]. Diese Vorschläge sind im Internet aber praktisch, mit Ausnahme von Overlay-Netz basierten Ansätzen (siehe Abschnitt 2.2.1), nicht verfügbar. Die Gründe dafür sind vielfältig. In erster Linie zeichnen aber wohl nicht technische Probleme, sondern eher politische bzw. wirtschaftliche Gründe dafür verantwortlich, da eine große Zahl an Netzbetreibern kooperieren und sich auf einen Standard einigen müsste.
Der in Abschnitt 2.2.3 beschriebene Ansatz des Application Layer Multicast eignet sich, um Multicast-Dienste praktisch zur Verfügung zu stellen. Da in diesem Ansatz aber die physikalische Netztopologie nicht berücksichtigt werden kann, führt dies unweigerlich zu Ressourcenverschwendung im Netz. Die Technologie der Programmierbaren Netze kann hier eine Verbesserung bringen. Durch flexible Netzdienste können Application Layer Multicast Overlay-Netze angeboten werden, die in einem Programmierbaren Netz bereitgestellt werden. Damit steht die notwendige Flexibilität zur Verfügung, um Multicast-Dienste anbieten zu können. Im Vergleich zu endsystembasiertem Application Layer Multicast, können diese Ansätze effizienter mit Ressourcen im Netz umgehen. Der von Tani [ST01] vorgeschlagene Netzdienst ist ein Vertreter der beschriebenen Application Layer Multicast Overlay-Netze in Programmierbaren Netzen. Aufbauend auf Unicast-Verbindungen zwischen Dienstmodulen auf programmierbaren Netzknoten wird ein Application Layer Multicast-Dienst angeboten.

Die exemplarisch präsentierten Dienste zeigen anschaulich die Verknüpfung von Overlay-Netzen mit den Programmierbaren Netzen. Programmierbare Netze eignen sich sowohl zur Unterstützung von existierenden Overlay-Netzen, als auch, um eigenständige, dynamische Overlay-Netze als Dienst anzubieten.
Der in dieser Arbeit vorgeschlagene flexible Fehlteroleranzdienst basiert ebenfalls auf Overlay-Netzen. Darüber hinaus kann der Dienst als Mischform der beschriebenen Kategorien charakterisiert werden. zum einen kann der Dienst vollkommen transparent gegenüber dem Benutzer im Netz ablaufen und somit generell den Netzbetrieb verbessern, zum anderen kann auf Wunsch direkt mit den Applikationen kooperiert werden.

2.5 Fehler und Fehlertoleranz in Systemen

In der Entwicklung von Kommunikationsnetzen ist der Umgang mit Fehlern seit jeher ein wichtiger Punkt. Kommunikationssysteme können zu den komplexesten Produkten gehören, die von Menschen hergestellt werden. So hat z.B. die Software des Vermittlungssystems “Elektronischen Wählsystems Digital“ (EWSD) einen Umfang von mehr als 45 Millionen Zeilen Programmcode [Tsa98]. Für das in dieser Arbeit betrachtete Internet kann aufgrund des heterogenen Aufbaus und ständigen Wandels nicht ohne weiteres eine entsprechende Zahl angegebenen werden. Offensichtlich herrscht aber auch auf diesem Gebiet große Komplexität. Diese macht es praktisch unmöglich, Systeme herzustellen die fehlerfrei sind. Es kann nur teilweise bzw. exemplarisch getestet werden, ob komplexe Systeme korrekt funktionieren. Zu

den möglichen Fehlern, die schon im Design von Systemen verborgen sein können, seien es nun z.B. Materialfehler oder Fehler in der Software, kommt zusätzlich noch falsche Bedienung von außen als weiterer Grund für Fehler hinzu.
Das Auftreten von Fehlern in Systemen ist eine Tatsache, der wir uns auch im täglichen Leben oft gegenüber sehen. Dies können relativ einfache Fehler sein, wie z.B. das Durchbrennen einer Lampe. Diesen Fehler kann man noch vollständig analysieren und verstehen. Auch sehr komplexe Fehler wie z.B. der Absturz eines Computerprogramms oder Stromausfälle in ganzen Landstrichen sind bekannt. Hier wird es allerdings schon schwieriger bzw. u.U. sogar unmöglich, den Fehler zu verstehen.
Die Behandlung von Fehlern ist ein wichtiger Bestandteil moderner Systeme. In diesem Abschnitt gehen wir zuerst auf die genauere Definition von Fehlern durch Fehlerbegriffe ein. Des Weiteren geben wir eine kurze Einführung in die Behandlung von Fehlern.

2.5.1 Fehlerbegriffe

Die gegebenen Fehlerbeispiele (Durchbrennen einer Lampe, Absturz eines Computerprogramms) zeigen eines der Grundprobleme im Umgang mit Fehlern von komplexen Systemen. Um Fehler beheben zu können, kann man versuchen, den Fehler vollständig zu verstehen und anschließend zu behandeln. Was bedeutet das aber jetzt im Detail?
Im Falle der Lampe wäre ein vollständiges Verständnis möglich. Eine Analyse würde ergeben, dass z.B. der Glühdraht aufgrund einer Materialermüdung durchtrennt ist. Eine mögliche Fehlerkorrektur wäre dann z.B. diesen Draht wieder zusammenzufügen. Aus praktischen Gründen würde dies jedoch nicht gemacht, sondern wohl eher die defekte Glühbirne durch eine neue ersetzt. Das Resultat ist in beiden Fällen identisch, der Fehler wäre behoben. Man sieht hier anschaulich, dass Fehler in Systemen auf unterschiedliche Art und Weise behoben werden können.
Für den Fall eines Fehlers in einem komplexeren System, wie z.B. dem Absturz eines Computerprogramms, könnte man u.U. ebenfalls versuchen, diesen Fehler vollständig zu analysieren und zu verstehen. Aufgrund der Komplexität ist dies aber u.U. praktisch nicht möglich. Es könnte sein, dass z.B. eine bestimmte Kombination von Eingaben in Kombination mit verschiedensten Interaktionen innerhalb der Maschine zum Fehler geführt haben. Nehmen wir weiter den Fall an, dass dieses Verhalten schwer reproduzierbar ist und der Fehler somit nur hin und wieder auftritt. Dann wäre eine Analyse äußerst aufwendig und u.U. sogar erfolglos. Für die Behandlung von Fehlern in komplexen Systemen ist daher eine genauere Betrachtung von Fehlern notwendig. In der Literatur [Bre01] werden daher Fehler durch drei *Fehlerbegriffe* beschrieben:

- Fehlerursache
- Fehlerzustand
- Versagen

Die *Fehlerursache* (engl. fault) ist die ermittelte Ursache für einen Fehler. Sie kann zum einen im System selbst liegen, z.B. durch Defekte von Komponenten, zum anderen gibt es Fehlerursachen, die außerhalb des Systems ihren Ursprung haben, z.B. eine falsche Bedienung eines Systems. Fehlerursachen werden durch Fehleranalysen ermittelt. Wie bereits

exemplarisch dargestellt, ist dies u.U. praktisch nicht ohne weiteres möglich bzw. aufwendig.
Mit *Fehlerzustand* (engl. error) wird ein Zustand eines Systems bezeichnet der vom gewollten, spezifizierten Systemzustand abweicht. Breitling gibt als einen möglichen Fehlerzustand z.B. einen falschen Wert in einer Datenbank während eines definierten Zustands eines Systems an. Die Ursache, warum es zu diesem Fehlerzustand gekommen ist, bleibt bei der Betrachtung von Fehlerzuständen außen vor. Im Beispiel wäre es somit möglich den Fehler zu beheben, indem man einfach den korrekten Wert in die Datenbank einträgt, und damit den Fehlerzustand korrigiert. Die eigentliche Ursache, die zu dem Fehlerzustand im System geführt hat, bliebe bei dieser Art Korrektur unberücksichtigt.
Das *Versagen* (engl. failure) von Systemen ist definiert als die Abweichung vom Verhalten des Systems gegenüber der Spezifikation. Das Versagen kann, ebenso wie ein Fehlerzustand, nur auftreten, wenn das System auch betrieben wird. Breitling beschreibt die Korrektheit eines Systems als die potentielle An- bzw. Abwesenheit von Versagen.

Weiter sind die kausalen Zusammenhänge der beschriebenen Fehlerbegriffe wichtig für das Entstehen von Fehlern. Eine *Fehlerursache* kann einen *Fehlerzustand* des Systems zur Folge haben. Dies ist aber nicht zwingend notwendig. So kann z.B. ein System fehlerhafte Komponenten haben. Werden diese Komponenten aber im Betrieb nicht benutzt, dann tritt u.U. kein *Fehlerzustand* auf.
Gleiches gilt für Fehlerzustände und Versagen. Ein *Fehlerzustand* kann zum *Versagen* des Systems führen. Dies ist aber ebenfalls nicht zwingend notwendig, wenn z.B. der Fehlerzustand, bevor es zu einem Versagen kommt, korrigiert wird.
Im Umkehrschluss muss das System aber bei einem Versagen vorher in einem Fehlerzustand gewesen sein. Dieser aufgetretene Fehlerzustand muss ebenfalls zwingend eine Fehlerursache haben.

2.5.2 Fehlermodelle

In Abschnitt 2.5.1 wird das Auftreten von Fehlern durch die Abfolge von Fehlerursache, Fehlerzustand und dem Versagen des Systems beschrieben. Dieses Systemversagen lässt sich im Allgemeinen [Bre01] durch vier Fehlermodelle charakterisieren:

- Crash-Versagen
- Fail-Stop-Versagen
- Omission-Versagen
- Byzantine-Versagen

Crash-Versagen beschreibt das willkürliche und vollständige Versagen eines Systems. Das System läuft fehlerfrei, plötzlich steht es vollständig still. Somit können Nachrichten weder in das System eingebracht noch aus dem System ausgegeben werden. Ein Versagen des Systems ist somit von außen nicht unmittelbar feststellbar.
Mit *Fail-Stop-Versagen* ist ein dem Crash-Versagen ähnliches Fehlermodell gemeint. Das System bleibt ebenfalls unmittelbar vollständig stehen. Im Unterschied zum Crash-Versagen

ist in diesem Fall aber noch eine Meldung an die Systemumgebung möglich, die dieses Versagen anzeigt. Das Versagen kann somit von außen unmittelbar festgestellt werden.
Von *Omission-Versagen* spricht man, wenn Verluste bei der Ausgabe eines Systems auftreten. Die Bandbreite reicht vom vollständigen Verlust aller Nachrichten nach außen bis zum Verlust nur einer einzigen Nachricht. Für den Fall des vollständigen Verlustes aller Nachrichten, ist von außen, ähnlich wie beim Fall des Crash-Versagens, der Fehler nicht unmittelbar feststellbar.
Die Fehlermodelle des Crash-Versagens und des Fail-Stop-Versagens beschreiben den Fehler eines Systems als vollständigen Stillstand. Das Fehlermodell des Omission-Versagens berücksichtigt nur die Beeinträchtigung der Kommunikation des Systems mit der Umgebung.
Das *Byzantine-Versagen* beschreibt ein Fehlermodell, in dem das System nicht notwendigerweise stehen bleibt. In diesem Fehlermodell wird ein chaotisches Verhalten des Systems modelliert. Damit ist gemeint, dass das System z.B. weiterläuft, aber nicht das tut, was es eigentlich soll. Das System läuft somit unkontrolliert ab.

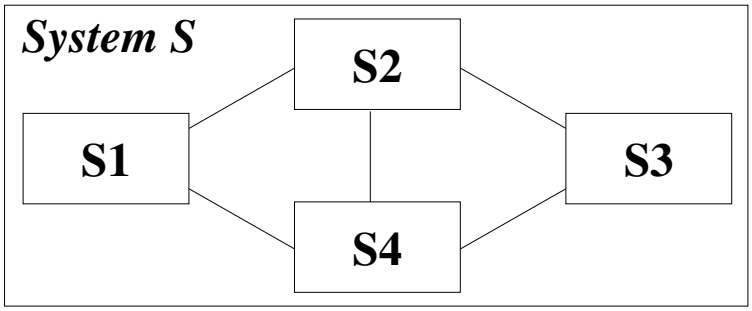

Abbildung 2.5: Unterteilung in Teilsysteme

Betrachtet man das Internet aus Systemsicht, so kann man für die Modellierung eines jeweiligen Versagens die beschriebenen unterschiedlichen Fehlermodelle heranziehen. In dieser Arbeit betrachten wir das Internet als eine Komposition aus Teilsystemen. Die in Abbildung 2.1 dargestellte Kommunikation zwischen Host A und Host B über das Internet kann durch das in Abbildung 2.5 dargestellte System S modelliert werden. Das System S besteht aus den Teilsystem (bzw. Subsystemen) S1 bis S4. S1 modelliert aus Abbildung 2.1 Host A mit Netz 2. S2 und S4 modellieren die Netze 1 und 4. Das Teilsystem S3 modelliert Netz 3 mit Host B.
Die dargestellten Fehlermodelle können nun auf die Teilsysteme angewandt werden. So wäre z.B. folgendes zusammengesetztes Fehlermodell des Systems S möglich:

- Die Teilsysteme S1, S3 und S4 funktionieren fehlerfrei.
- Für das Teilsystem S2 nehmen wir Crash-Versagen an.

Mit diesem relativ einfachen Fehlermodell wäre z.B. der Fall beschrieben, dass Netz 1 aus Abbildung 2.1 ausfällt.

Das in Abbildung 2.5 präsentierte Systemmodell des Internets ist stark vereinfacht. Für die detaillierte Betrachtung der Abläufe und die Modellierung möglicher Fehler ist eine Verfeinerung des Systemmodells notwendig. So kann man ein Versagen zusätzlich noch zeitlich

betrachten [Gör89]. Görke führt als Verfeinerung des Versagens die Modelle der permanenten, transienten oder wiederkehrenden Fehler ein.
Die vorliegende Arbeit modelliert Kommunikationsnetze durch zusammengesetzte Fehlermodelle mit Crash-Versagen in Subsystemen. Augenmerk wird auf die Behandlung von quasi-permanenten und transienten Fehlern gelegt.

2.5.3 Fehlertoleranz

Die in Abschnitt 2.5.2 dargestellte Systemsicht (in der Kombination mit Fehlermodellen), ist zusammen mit den in Abschnitt 2.5.1 dargelegten Fehlerbegriffen eine Grundlage für die Behandlung von Fehlern in Kommunikationsnetzen bzw. im Internet.
Unter Fehlertoleranz versteht man die Fähigkeit eines Systems, auch mit einer begrenzten Zahl fehlerhafter Subsysteme die spezifizierte Funktion zu erfüllen. Die beschriebenen Fehlermodelle, in Kombination mit den Fehlerbegriffen, stellen die Grundlage dar, um Fehlertoleranzmechanismen allgemein in Systemen bzw. in unserem Fall in Kommunikationsnetzen zu beschreiben.

Fehlertoleranz in Systemen wird in der Literatur [Gör89] als Zusammenspiel von drei Modulen beschrieben:
Zum einen müssen Fehler bzw. Fehlerzustände in Systemen durch eine *Fehlerdiagnose* festgestellt werden können. Für festgestellte Fehler muss es eine Vorgehensweise zur *Fehlerbehandlung* geben. Die Behandlung von Fehlern bzw. deren Behebung beruht letztlich auf dem *Vorhandensein von Redundanz*.

Ein Fehlertoleranzmechanismus beginnt daher mit der *Diagnose* eines Fehlers. In komplexen Systemen spielt dabei die beschriebene Systemsicht in Kombination mit Fehlerbegriffen eine große Rolle. So ist es zunächst unerheblich, warum ein Fehler aufgetreten ist. Wie in Abschnitt 2.5.1 beschrieben, kann eine Analyse des Fehlers und das Finden der Fehlerursache zeitlich aufwendig sein. Ziel von Fehlertoleranzmechanismen sollte es aber sein, Fehler (bzw. Fehlerzustände) im System schnell zu beheben. In einem separaten Schritt kann natürlich eine Analyse erfolgen, die u.U. in einer Reparatur z.B. durch den manuellen Austausch von defekten Komponenten mündet.
Die Fehlerdiagnose des Systems stellt das Versagen eines Teilsystems fest. Dieses Versagen bringt das Gesamtsystem in einen Fehlerzustand.
An diesem Punkt setzt die *Fehlerbehandlung* des Systems ein. Ziel der Fehlerbehandlung ist, ein System aus dem Fehlerzustand wieder in einen regulären Zustand zu überführen, um ein Versagen des Systems zu verhindern.
Görke beschreibt folgende Möglichkeiten zur Fehlerbehandlung in Systemen:

- Fehlerbehebung (engl. error recovery)
  - Rückwärtsfehlerkorrektur (engl. backward error recovery)
  - Vorwärtsfehlerkorrektur (engl. forward error recovery)
- Fehlerkompensation (engl. error compensation)
  - Fehlermaskierung (engl. fault masking)

– Fehlerkorrektur (engl. error correction)

Rückwärtsfehlerkorrektur bzw. *Vorwärtsfehlerkorrektur* sind Methoden der Fehlerbehandlung, die ein System von einem Fehlerzustand in einen korrekten Zustand überführen, der schon einmal aufgetreten ist, oder in einen bekannten zukünftigen korrekten Zustand. Dabei wird jeweils die gegenwärtig vom System ausgeführte Funktion abgebrochen und das System anschließend in einen korrekten Zustand gebracht. Im System muss daher diese Information über vorhergehende bzw. zukünftige korrekte Zustände vorhanden sein. Diese Information ist für (theoretische) Systeme, in denen keine Fehler auftreten können, redundant. Für praktische, mit Fehlern behaftete Systeme ist diese Information notwendig, um Fehlertoleranz in das System zu integrieren. Wir sehen also die Notwendigkeit von Redundanz in fehlertoleranten Systemen. Ein Beispiel für Fehlerbehebung wäre der Neustart eines Rechenprozesses, nachdem für diesen ein Fehlerzustand detektiert wurde.

Die Methoden der *Fehlerkompensation* versuchen generell ein Versagen, das in einem Subsystem aufgetreten ist, gegenüber dem übergeordneten System, das evtl. selbst wieder als Subsystem in einem größeren System agiert, zu maskieren. So ist es für die Fehlermaskierung charakteristisch, identische Komponenten mehrfach und damit redundant im System zu integrieren. Kommt es zum Versagen einer der Komponenten, dann kann durch redundante Komponenten die gewünschte Funktionalität durch eine zusätzliche Logik gewährleistet werden. Das übergeordnete System bemerkt vom Versagen der Komponente nichts.
Ein Beispiel für die klassische Fehlermaskierung in der Telekommunikation ist *Ersatzschalten mit vordefinierten Wegen und Kapazitäten* (engl. protection switching). Dabei werden mögliche Ersatzwege und deren Kapazität bereits vor einem Ausfall konfiguriert [Ebe02].

Eine weitere Möglichkeit der Fehlerkompensation ist die *Fehlerkorrektur*. Hier können Fehler, die in einem Subsystem aufgetreten sind, durch das übergeordnete System korrigiert werden. Die nötige Redundanz liegt nicht in zusätzlichen Komponenten sondern in den bearbeiteten Daten. Ein Beispiel hierfür sind Forward Error Correction Codes.

Kommunikationsnetze bestehen allgemein aus Hardware- und Software-Komponenten. So kann die Fehlerursache, wenn man Einflüsse von außen nicht berücksichtigt, in der Hardware oder Software zu finden sein. Der in der vorliegenden Arbeit präsentierte Fehlertoleranzmechanismus unterscheidet nicht zwischen Hardware- oder Softwarefehlern, da Grundlage dieser Arbeit nicht die Fehlerursachen, sondern lediglich die Fehlerzustände in den höheren Schichten sind. In welchen Schichten letztlich die Fehlerursachen zu finden sind, wird dabei nur am Rande betrachtet.

2.6 Zusammenfassung

In diesem Kapitel wurden die wesentlichen Grundlagen für diese Arbeit präsentiert. Zu Beginn wird der Aufbau von IP-basierten Netzen bzw. des Internets dargestellt. Danach stellen wir verschiedene Ansätze von Overlay-Netzen vor. Außerdem geben wir einen kurzen Überblick über Programmierbare Netze und Netzdienste. Zum Schluss gehen wir noch auf Fehler und deren Behandlung in Systemen ein. Als grundlegend für die Arbeit sind folgende Aspekte anzuführen:

- Das Internet besteht aus einer Vielzahl an administrativ unabhängigen, lose vermaschten Netzen. Dadurch ist prinzipiell die Möglichkeit gegeben, unterschiedliche kanten- und knotendisjunkte Pfade für den Datentransport zwischen Kommunikationspartnern zu finden.
- Die Abbildung der Schichtenarchitektur auf Softwarekomponenten eines Betriebssystems stellt die grundsätzlichen Anforderungen für die Plattform und Architektur des vorgeschlagenen Netzdienstes dar.
- Die Mächtigkeit von Overlay-Netzen wurde dargelegt. Insbesondere der Ansatz von P2P-Overlay-Netzen spielt eine wichtige Rolle für diese Arbeit.
- Das Konzept der Programmierbaren Netze eignet sich als Dienstplattform für Fehlertoleranzdienste im Netz.
- Die gegebene Einführung in das Gebiet von Fehlern in Systemen und deren Behandlung ist die Grundlage für die nötige Abstrahierung, um die Funktionalität des beschriebenen Fehlertoleranzdienstes zu erläutern.

Durch Kombination von Programmierbaren Netzen mit dem Konzept der P2P-Netze wird Fehlertoleranz als Netzdienst vorgeschlagen. Durch die flexible Grundlage kann dieser Netzdienst an Bedürfnisse von Applikationen angepasst werden. Dies ist im heterogenen und sich darüber hinaus ständig ändernden Umfeld des Internets von Vorteil. Im nächsten Kapitel gehen wir detailliert auf den Stand der Technik ein.

Kapitel 3

Stand der Technik

Dieses Kapitel grenzt das Gebiet der Arbeit weiter ein. Zuerst erfolgt ein kurzer Überblick über bestehende niederschichtige Fehlertoleranzmechanismen, die sowohl in homogenen als auch in heterogenen Netzarchitekturen verankert sind. Danach geben wir einen Einblick in höherschichtige Fehlertoleranzmechanismen, die auf Overlay-Netzen (siehe Abschnitt 2.2) aufbauen. Zusätzliches Augenmerk schenken wir Verfahren des Datentransports in Overlay-Netzen. Neben den bisher angesprochenen netzbasierten Fehlertoleranzmechanismen betrachten wir weiter applikationsabhängige Fehlertoleranzmechanismen, die in den kommunizierenden Endsystemen basiert sind.
Des Weiteren gibt dieses Kapitel einen Überblick über vorgeschlagene Architekturen von programmierbaren Knoten bzw. Netzen. Der in dieser Arbeit präsentierte Fehlertoleranzdienst basiert auf Programmierbaren Netzen. Wesentliche Eigenschaften dieses Dienstes hängen von den Möglichkeiten der zur Verfügung stehenden programmierbaren Elemente des Netzes ab.

3.1 Überblick: Fehlertoleranz im Internet

Die in [Ise99] beschriebenen Hierarchiestufen für Fehlertoleranz in Kommunikationsnetzen sind in Abbildung 3.1. detaillierter dargestellt. Der von Iselt allgemein für Kommunikationsnetze beschriebene Ansatz stellt Kommunikationsnetze aus der Sicht der Fehlertoleranz in vier Ebenen dar. Er betrachtet die Baugruppenebene, aus der sich Netzelemente (z.B. Router) zusammensetzen. Diese zwei Ebenen fassen wir in dieser Arbeit zu einer Ebene, den *Netzelementen*, zusammen. Im Folgenden wird diese Ebene nicht weiter betrachtet. Darüber führt Iselt die (eine) Netzebene an, die sich aus Netzelementen zusammensetzt. Diese Ebene übernehmen wir in unserem Modell, unterteilen sie aber in eine *domänenbegrenzte Netzebene* und eine *Internetschicht*. Die Ebenen über der Netzebene subsumiert Iselt unter dem Begriff der Anwendungsebene. Hier teilen wir ebenfalls auf in eine *Overlay-Netzebene* und eine *Applikations-Ebene*. Die detaillierte Betrachtung der höheren Ebenen ist Grundlage dieser Arbeit. Wir fokussieren uns auf die Unterstützung von netzbasierten Applikationen und betrachten die Applikationsebene, die Ebene der Overlay-Netze und die Internetschicht. Die domänenbegrenzte Netzebene berücksichtigen wir nur am Rande.

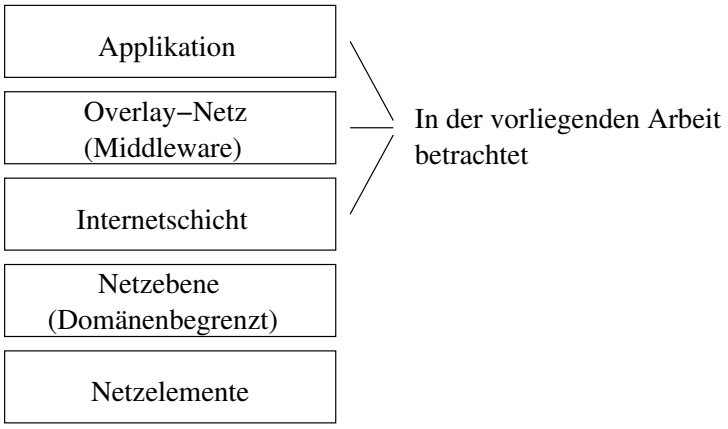

Abbildung 3.1: Hierarchiestufen für Fehlertoleranz im Internet [TB02]

Im Folgenden wird der Stand der Technik von Fehlertoleranzmechanismen in diesen Ebenen dargestellt.

3.2 Fehlertoleranz in Netzebene und Internetschicht

In diesem Abschnitt betrachten wir exemplarisch Fehlertoleranzmechanismen in domänenbegrenzten Netzen und in der Internetschicht. Gemeinsames Merkmal dieser Mechanismen ist die *Fehlerkompensation* durch *Ersatzschalten*. Ziel ist es, den Datentransport in der jeweiligen Ebene von einem Netzrand zum anderen sicherzustellen. Die am Netzrand basierten Applikationen werden bei diesen Mechanismen nicht oder nur aggregiert berücksichtigt.
Prinzipiell sind zwei Methoden des Ersatzschaltens definiert [Ebe02]:

Beim *Protection Switching* sind Ersatzwege bzw. -kapazitäten bereits vordefiniert. Somit kann bei einem Ausfall ohne zusätzliche Konvergenzzeit auf einen alternativen Pfad umgeschaltet werden.
Das Verfahren der *Restauration* greift nicht auf reservierte Ersatzkapazitäten zurück. Im Falle eines Ausfalls muss zunächst ein möglicher alternativer Pfad ermittelt werden. Erst nach erfolgter Konfiguration kann auf einen Ersatzpfad umgeschaltet werden. Die für den Ablauf des Restaurationsverfahrens benötigte Zeit nennt man Konvergenzzeit. Bis die neu berechneten Pfade im Netz zur Verfügung stehen, werden Datenpakete in der Regel verworfen.

Im Folgenden bezeichnen wir der Einfachheit halber *Protection Switching* als *Ersatzschalten*. Für den Fall des *1+1 Ersatzschaltens* wird der alternative Pfad parallel schon vor einem möglichen Ausfall genutzt, und die zu übermittelnden Daten redundant übertragen. Die Fälle des *1:1* bzw. *1:N Ersatzschaltens* konfigurieren einen alternativen Pfad, benutzen diesen aber erst nach einem Ausfall.

3.2.1 Integrierte Fehlertoleranz in Netzarchitekturen

In [Aut03] werden mehrschichtige optische Transportnetze untersucht. Die Arbeit gibt einen Überblick über den *Asynchronous Transfer Mode* (ATM), die *Synchrone Digitale Hierarchie* (SDH), *Optische Transport Netze* (OTN) und das *Multiple Protocol Label Switching* (MPLS). Insbesondere werden verschiedene Multilayer-Ansätze vorgestellt, die diese Technologien kombinieren, um mehrschichtige IP-Transportnetze zu realisieren.
Die Arbeit gibt einen Überblick über die veröffentlichten und teilweise standardisierten Fehlertoleranzmechanismen in den jeweiligen Technologien. So werden die für ATM standardisierten unidirektionalen bzw. bidirektionalen Ersatzschaltverfahren dargelegt. Für SDH werden ebenfalls vorgeschlagene und bereits standardisierte Ersatzschalt- bzw. Restaurations-Verfahren präsentiert. Auch rein optische Verfahren werden in einem Überblick erklärt. Für das in letzter Zeit populär gewordene MPLS werden ebenfalls entsprechende Fehlertoleranzmechanismen dargestellt.
Diese in den jeweiligen Technologien bzw. Schichten ablaufenden Fehlertoleranzmechanismen werden in sog. *Multilayer-Resilience*-Mechanismen integriert, um die Fehlertoleranz insgesamt zu verbessern. Der Vorschlag von Autenrieth bezieht zusätzlich noch die über der IP-Schicht laufenden Applikationen bzw. Dienste mit ein und schlägt eine *Resilience Differentiated* Quality of Service (QoS) vor. So werden bei einem Ausfall im Transportnetz Datenströme durch die Fehlertoleranzmechanismen unterschiedlich behandelt, gemäß eingeführter QoS-Parameter. Letztlich ist auch hier das Ziel, die Fehlertoleranz insgesamt zu verbessern. Kernidee der Arbeit ist die Differenzierung von IP-Verkehr in Fehlertoleranzkategorien. So werden Datenströme in unterschiedliche Fehlertoleranzkategorien aggregiert und entsprechend behandelt.

Gemeinsamkeit dieser Vielzahl an teilweise mehrschichtigen Fehlertoleranzmechanismen ist die Sichtweise auf ein durchgängiges, homogenes Netz. In domänenbegrenzten Netzen ist das üblicherweise der Fall. Das Internet besteht aber aus einer Vielzahl an heterogenen Netzen. In unserem Beispiel in Abbildung 2.1 könnte es z.B. sein, dass in Netz 1 ein mehrschichtiger Fehlertoleranzmechanismus integriert ist, wie von Autenrieth skizziert. Die übrigen Netze könnten aber auf einer anderen Technologie basieren bzw. aufgrund unterschiedlicher administrativer Domänen nicht in den Fehlertoleranzmechanismus von Netz 1 eingebunden sein. Somit wäre die Durchleitung der Daten durch den lokalen Fehlertoleranzmechanismus zwar in Netz 1 geschützt. Kommt es aber zu einem Versagen, auf den dieser Mechanismus keinen Einfluss hat, dann ist die Übertragung der Daten nicht geschützt. Denkbar wären hier konstante Ausfälle von Single Points of Failures, z.B. durch einen Terroranschlag auf der einen Seite der Skala, aber auch kleinere, temporäre Ausfälle, z.B. durch menschliches Versagen bei einem Software-Update einer Netzkomponente auf der anderen Seite der Skala. Im Internet ist eine Vielzahl an Netzen verbunden. Den skizzierten möglichen Ausfällen, die nicht durch Fehlertoleranzmechanismen innerhalb administrativer Domänen behoben werden können, wird durch Fehlertoleranzmechanismen in der Internetschicht begegnet. Dies wird im nächsten Abschnitt näher erläutert.

Die kurz angeführten Fehlertoleranzmechanismen in domänenbegrenzten Netzarchitekturen sind schon seit langer Zeit ein breites Forschungsgebiet. Die beschriebenen Ansätze befinden sich mittlerweile auf hohem Niveau und sind z.T. schon in der Praxis umgesetzt. Der in der vorliegenden Arbeit vorgeschlagene, netzübergreifende Fehlertoleranzdienst steht nicht

in Konkurrenz zu den angegebenen Mechanismen.
Diese Arbeit setzt in zwei wesentlichen Punkten an: Zum einen gehen wir auf die, im nächsten Abschnitt skizzierten, Problem der netzübergreifenden Fehlertoleranz ein. Zum anderen unterstützt der beschriebene Fehlertoleranzdienst direkt die spezifischen Anforderungen der am Netzrand allozierten Applikationen. Wir berücksichtigen in der in Abbildung 3.1 angeführten Hierarchie des Internets aus der Sicht der Fehlertoleranz die Schichten unterhalb der Internetschicht nicht. Aus diesem Grund ergänzt sich die vorliegende Arbeit mit den in diesem Abschnitt beschriebenen Fehlertoleranzmechanismen.

3.2.2 Internetschicht

Das Internet besteht, wie in Abbildung 2.1 exemplarisch skizziert, aus einer Vielzahl von eigenständigen und administrativ unabhängigen Domänen bzw. Netzen. Diese Domänen bezeichnet man als *Autonome Systeme (AS)*. Wie in Abschnitt 2.1.1 dargestellt, sind AS im Internet lose vermascht und leiten Daten von einem Nachbar-AS zu einem anderen Nachbar-AS weiter.
Um diese Weiterleitung zu ermöglichen, ist ein *Interdomain Routing-Protokoll* notwendig, das den AS topologische Informationen, wie bestimmte Netze erreicht werden können, zur Verfügung stellt. Gegenwärtig wird hierfür im Internet das Border Gateway Protokoll (BGP) der Version Nr. 4 [RL95] verwendet. Mittels BGP werden im Internet Informationen an alle AS verteilt, wie bestimmte Netze zu erreichen sind.
Von Beginn an wurde das Internet fehlertolerant konzipiert. Mittels eines *Update*-Mechanismus ist mit BGP ein Restaurationsverfahren möglich. Im Falle eines Versagens initiiert BGP ein verteiltes Rerouting. Neue Routen werden berechnet. Nach einer Konvergenzzeit, bis die neuen Routen im Netz verteilt sind, steht die Konnektivität wieder zur Verfügung.
Die für die Restauration benötigte Zeit, für die in der Regel keine Konnektivität zur Verfügung steht, ist aber weit über dem Toleranzbereich einiger Applikationen. In einer zwei Jahre dauernden Studie [LABJ01], die BGP Restaurationsverfahren im Internet untersucht, wird die Konvergenzzeit der Restauration in der Größenordnung von einigen Minuten bis zu einigen zehn Minuten angegeben. Die benötigte Zeit ist stark vom konkreten Einzelfall abhängig. Für diese Arbeit von Relevanz ist allein schon die gemessene Größenordnung der Konvergenzzeit. Selbst Ausfälle von nur einigen Sekunden wären für einige Applikationen nicht mehr tolerierbar. Die gemessenen Konvergenzzeiten liegen aber um mindestens eine Größenordnung darüber. Somit kann das Versagen in der Internetschicht selbst für Applikationen, die einen relativ niedrigen Anspruch an die Fehlertoleranz bei der Datenübertragung haben (z.B. durch Puffermechanismen geschütztes, unidirektionales Streaming), zum Versagen auf der Applikationsebene führen.

Die Fehlertoleranz von BGP wird durch zahlreiche Vorschläge verbessert. Ausfälle im BGP können durch Fehler im verteilten Routing auftreten. Obwohl kein physikalischer Fehler auftritt, wie z.B. ein Linkausfall, kann es sein, dass z.B. durch falsche Konfiguration eines BGP-Routers oder durch eine gezielte externe Attacke gewisse Routen durch ASe nicht zur Verfügung stehen. Dieser Art von Ausfällen begegnen Vorschläge, die das Verhalten von Inter Domain Routing-Protokollen bei internen Fehlern verbessern. In [Zha02] wird z.B. das *Fault-Tolerant Border Gateway-Protokoll* vorgeschlagen. Ansätze dieser Art können die

Häufigkeit von Ausfällen reduzieren. Auf das Verhalten von BGP bei einem Ausfall (und somit auf die Konvergenzzeit) haben sie jedoch keinen Einfluss.
Dem Problem der langen Konvergenzzeit der Restaurationsverfahren im BGP stehen andere Vorschläge gegenüber. So schlagen Bless et al. ein *Fast Scoped Rerouting for BGP (FASRO)* [BLSZ03] vor. Zum einen unterdrückt FASRO unnötige bzw. falsche *update* Nachrichten im BGP. Dadurch wird, ähnlich wie im Falle des Fault-Tolerant Border Gateway-Protokolls, die Häufigkeit von Ausfällen reduziert. Zum anderen benutzt die FASRO-Architektur alternative Pfade. Ist ein Ausfall detektiert, werden die Autonomen Systemen in unmittelbarer Nachbarschaft zum fehlerhaften System, über den Ausfall informiert. Durch die anliegenden AS wird ein alternativer Pfad berechnet und konfiguriert. Dieses lokale Restaurationsverfahren bietet gegenüber reinem BGP den Vorteil, dass die Nachricht einer neu berechneten Route nicht global verteilt werden muss. Die FASRO-Architektur begrenzt die Verbreitung der Nachricht über eine neue alternative Route auf die AS in der Nachbarschaft, und stellt den alternativen Pfad bereit. In der beschriebenen Testumgebung aus 16 Autonomen Systemen kann mittels FASRO die Konvergenzzeit im Vergleich zu reinem BGP auf 40% reduziert werden. Setzt man diese mögliche Verbesserung in Relation zu den in [LABJ01] gemessenen Konvergenzzeiten, wird im Falle eines BGP-Ausfalls die FASRO-Konvergenzzeit ebenfalls in der Größenordnung von Minuten liegen.

Während die beschriebenen Mechanismen Ausfälle innerhalb des Internet behandeln, gibt es auch Vorschläge, die den Netzzugang für Endsysteme fehlertolerant gestalten. Im regulären Fall ist ein LAN durch ein bestimmtes Gateway mit dem Provider bzw. dem Internet verbunden. Das Gateway ist aus Sicht der Fehlertoleranz dann ein sog. *Single Point of Failure*. Kommt es im Gateway zu einem Versagen, dann ist sämtliche Kommunikation zwischen dem angeschlossenen LAN und dem Internet unterbunden.
Das Prinzip des *Multi-Homing* [Hus01] schließt ein LAN durch mindestens zwei unterschiedliche Provider (bzw. AS) an das Internet an. Somit stehen den Endsystemen im betreffenden LAN mehrere Gateways in das Internet zur Verfügung. Fällt eines der Gateways aus, kann ein anderes benutzt werden. Für den *Upstream*, d.h. für die Datenpakete mit Ursprung aus dem LAN und Ziel im Internet, ist die Wahl des entsprechenden Gateways über verschiedene Mechanismen möglich (z.B. [LCML98, KWW+98]). Da für den Fall des Upstreams nur die Endsysteme bzw. die im LAN konnektierten Gateways betroffen sind, und somit kein weiterer Router konfiguriert werden muss, ist die Umschaltung zwischen den Gateways relativ schnell zu bewerkstelligen. Für den Fall des Upstreams kann somit ein schnelles Ersatzschalten realisiert werden.
Betrachtet man aber in diesem Kontext den Downstream, d.h. die Datenpakete mit Ursprung aus dem Internet und Ziel im LAN, ist der Fall komplexer. Mittels BGP würden diese Datenpakete auf der eingestellten Route zum ursprünglichen, defekten Gateway transportiert werden. Somit ist eine Änderung in den BGP-Routing-Tabellen notwendig, damit die Datenpakete aus dem Internet zum alternativen, funktionierenden Gateway transportiert werden. Die Route des betroffenen LAN muss in die entsprechenden BGP-Routing-Tabellen eingetragen werden, was laut Huston auch einer der Gründe für die rasch wachsende Größe von BGP-Routing-Tabellen ist. Damit haben wir für den Fall des Downstreams ebenfalls das schon beschriebene, langwierige Restaurationsverfahren des BGP.
Multi-Homing ist mittlerweile weit verbreitet. Zusätzlich zu dem beschriebenen Fehlertoleranzmechanismus ist auch ein *Traffic Engineering* möglich, bei dem der Datenverkehr eines

LAN auf entsprechende Gateways aufgeteilt wird. Aus Sicht der Endsysteme ist Multi-Homing eine gute Möglichkeit, sich gegen einen Ausfall des Internet Providers zu schützen. Insbesondere die in der Regel länger andauernden Ausfälle von Gateways können damit umgangen werden. Die für den Downstream notwendige Änderung in den BGP-Routing-Tabellen führt zum angeführten BGP-Restaurationsverfahren mit langer Konvergenzzeit. Aufgrund der langen Konvergenzzeit, und dem damit verbundenen längeren Verlust der Konnektivität, wird Multi-Homing in der vorliegenden Arbeit nicht weiter betrachtet. Ausfälle von Gateways, die Endsysteme an das Internet anschließen, werden daher im Folgenden nicht weiter berücksichtigt.

3.3 Fehlertoleranz durch Overlay-Netze

Die in Abschnitt 3.2 dargestellten Fehlertoleranzmechanismen in domänenbegrenzten Netzen und in der Internetschicht stellen die Fehlertoleranz des Internets gegenüber Applikationen auf eine solide Basis. Es stehen Ersatzschaltmechanismen zur Verfügung, die Fehler innerhalb domänenbegrenzter Netze schnell beheben bzw. maskieren können.
Treten Fehler auf, die nicht innerhalb der domänenbegrenzten Netze behebbar sind, können die beschriebenen Restaurationsverfahren innerhalb der Internetschicht nach einer Konvergenzzeit wieder für einen fehlerfreien Zustand sorgen. Durch diese *selbstheilende* Charakteristik des Internets ist sichergestellt, dass nach einem Ausfall die Funktionalität mit einer gewissen Verzögerung wiederhergestellt werden kann. Ein Problem in diesem Zusammenhang ist die angeführte mögliche lange Dauer der Konvergenzzeit. Für diesen Zeitraum steht in der Regel keine Konnektivität der Kommunikationspartner am Netzrand zur Verfügung. Aus diesem Grund wird intensiv daran gearbeitet diese Konvergenzzeit zu verkleinern und somit die Fehlertoleranz des Gesamtsystems zu verringern. Ansätze wie z.B. FORSA (siehe Abschnitt 3.2.2) können die Konvergenzzeit zwar verkleinern. Die absolute Größenordnung liegt aber auch dann noch in einem Bereich von Minuten, die von einigen Applikationen nicht toleriert werden kann. Dies ist in erster Linie in der hierarchielosen Architektur des Internets begründet.

Aus diesem Grund wurden in letzter Zeit einige Ansätze vorgeschlagen, die Overlay-Netze installieren, mit denen Fehler im Netz maskiert werden. In Abschnitt 2.2 wurde das Prinzip der Overlay-Netze, die zwischen der Internetschicht und der Applikationsebene angesiedelt sind, kurz erläutert. In diesem Abschnitt diskutieren wir zwei unterschiedliche Ansätze von Fehlertoleranz mit Overlay-Netzen: Zuerst Overlay-Netze, die allgemein den Datentransport im Netz gegenüber Ausfällen sichern. Anschließend Overlay-Netze, die im speziellen auf die Applikationen bzw. die zu transportierenden Daten mit eingehen, und somit den *Content* berücksichtigen.

3.3.1 Transportschichtbasierte Overlay-Netze

Das DETOUR-Projekt [SAA+99] ist eine der ersten Architekturen, die ein Overlay-Netz auf der Internetschicht installiert, um die Performanz von verteilten Applikation im Internet zu

verbessern. Ziel des Projektes sind in erster Linie die Reduzierung von Verzögerungszeiten für den Datentransport und die Verringerung von Paketverlusten. Der Hauptbeitrag des Projektes ist die netzübergreifende Bereitstellung von Quality of Service (QoS) Eigenschaften im Internet durch ein Overlay-Netz. Die Architektur sieht DETOUR-Router am Netzrand vor, die über IP-in-IP Tunnelmechanismen miteinander verbunden sind. Für die, durch das DETOUR-Overlay-Netz bereitgestellten, virtuellen Pfade werden QoS-Metriken bestimmt: Verzögerung, Verlustrate und zur Verfügung stehende Übertragungsrate. Basierend auf diesen Metriken wird durch das Overlay-Netz ein QoS-Routing zur Verfügung gestellt.
Fokus der DETOUR-Architektur ist klar die Optimierung von QoS-Eigenschaften. Des Weiteren ist angedacht, die DETOUR-Architektur ebenfalls zur Verbesserung der Fehlertoleranz zu nutzen. So ist in der DETOUR-Architektur ein *Multipath-Routing* möglich, d.h. ein Datenstrom kann auf unterschiedlichen Wegen durch das Overlay-Netz geleitet werden. Dieses Prinzip kann zur Umgehung von Ausfällen im Netz genutzt werden.

Das DETOUR-Projekt fokussiert sich auf den fehlerfreien Fall und verbessert QoS-Eigenschaften. Das *Resilient Overlay Networks* (RON) Projekt [ABKM01] baut auf dem Ansatz des DETOUR-Overlay-Netzes auf, und fokussiert sich auf die Verbesserung der Fehlertoleranz. RON ist die grundlegende Arbeit auf dem Gebiet der fehlertoleranten Overlay-Netze. Internet-Router werden zu sog. *RON-Clients* erweitert. In einem Application Layer Overlay Network (ALON) sind RON-Clients durch virtuelle Pfade verbunden. Hervorzuheben sind drei wesentliche Aspekte:

- Ein aus RON-Clients bestehendes ALON, ist innerhalb der Overlay-Schicht durch virtuelle Pfade vollvermascht.
- Innerhalb des ALON läuft ein Link-State Routing-Protokoll.
- Das RON wird intransparent an betreffende Endsysteme angeschlossen.

Grundsätzlich hat jeder RON-Client in einem N-Knoten Overlay-Netz, N-1 virtuelle Verbindungen zu den übrigen RON-Clients. Es handelt sich somit um ein durch virtuelle Pfade, vollvermaschtes Overlay-Netz. Der Aufbau eines RON ähnelt dem dynamischen Aufbau von unstrukturierten P2P-Netzen (siehe Abschnitt 2.2.4). Einem neu instanziierten RON-Client muss die Adresse eines schon bestehenden RON-Client beim Start übergeben werden. Von diesem Knoten bekommt er die Adressen von allen weiteren Teilnehmern des u.U. schon bestehenden RON. Anschließend werden zu allen bekannten RON-Clients Verbindungen aufgebaut. Die im RON aufgebauten virtuellen Pfade zwischen den RON-Clients werden in einem regelmäßigen Intervall überprüft.
Diese Überprüfung ist die Basis für ein Link-State Routing-Protokoll das innerhalb des RON Daten über virtuelle Pfade routet. Zwei über das RON kommunizierende Endsysteme sind jeweils über einen Ingress und Egress RON-Client an das RON angeschlossen. Wird im RON ein Ausfall zwischen den zwei betreffenden RON-Clients bemerkt, wird der Datenstrom über eine alternative Route im RON geroutet. Dabei wird durch das Link-State Routing-Protokoll ein dritter RON-Client benutzt, zu dem jeweils eine Verbindung zu den beiden Ingress bzw. Egress RON-Clients besteht. Das schon in der DETOUR-Architektur vorgeschlagene Multipath Routing in Overlay-Netzen, wird ebenfalls durch diesen Mechanismus realisiert.
Die Anbindung von Endsystemen bzw. die Benutzung des RON, erfolgt ähnlich der Benutzung von Proxies. Endsysteme senden ihre Daten direkt an einen RON-Client in ihrer Nähe. Im RON-Client werden die ankommenden Datenpakete dann enkapsuliert. Innerhalb

des RON erfolgt die Weiterleitung der Daten, UDP-basiert, mit einem zusätzlichen RON-Header. Folglich werden Datenpakete durch das RON vergrößert. Hat ein ankommendes Datenpaket aber schon die maximal mögliche MTU (im Falle von Ethernet 1500 Byte), wird eine MTU-Discovery ausgelöst. Schließlich muss das Endsystem die maximale Paketgröße entsprechend reduzieren. Durch diesen Mechanismus wird eine Fragmentierung von Paketen innerhalb des RON vermieden. Somit ist die transparente Benutzung eines RON für die betreffenden Endsysteme nicht möglich. Bestehende Applikationen müssen an das RON angepasst werden.

Durch den vollvermaschten Aufbau ist RON spezialisiert auf eine geringe Anzahl von kommunizierenden Endsystemen. Die Architektur skaliert gut bis zu einer Zahl von etwa 20 RON-Knoten. Dies liegt in erster Linie in der hierarchielosen Vollvermaschung begründet. Durch die geringe Zahl an möglichen RON-Clients ist die Anwendung von RON ebenfalls begrenzt. So sind Anwendungen möglich zwischen bekannten Kommunkationspartnern, wie dies z.B. in einem Intranet von global agierenden Firmen der Fall ist. In diesem Fall würden die jeweiligen Kommunikationspartner schon vor dem Aufsetzen der RON-Architektur feststehen. Man könnte exemplarisch einen zentralen Intranet-Server annehmen, auf den aus weltweit verteilten Unternehmensteilen über das Internet zugegriffen werden soll. Durch die RON-Architektur könnte man ein derartiges Kommunikationsszenario gegen Netzausfälle schützen.
Abgesehen von der mangelnden Skalierbarkeit hat RON noch einen weiteren Nachteil. Durch den intransparenten Ansatz werden Datenpakete, die durch RON geschützt werden sollen, immer durch die RON-Architektur geleitet. Diese Datenpakete werden im Ingress RON-Client entsprechend enkapsuliert und an den Egress RON-Client weitergeleitet. Auch für den Fall, dass u.U. lange Zeit kein Ausfall im Netz vorkommt, findet der Datentransport auf beschriebene Weise statt.
Dies bedeutet einen konstanten Overhead, der im fehlerfreien Fall unnötig ist: Datenpakete könnten im fehlerfreien Fall auch außerhalb des RON, und somit ohne zusätzliche Header, über das Internet transportiert werden könnten. Bei seltenen Ausfällen ist eine durch RON geschützte Kommunikation somit wenig effizient.

Die RON-Architektur ist grundlegend für die Entwicklung von Overlay-Netzen zur Verbesserung der Fehlertoleranz von verteilten Applikationen. Darauf aufbauend gibt es weitere Arbeiten, wie z.B. der Vorschlag von Zhao et al. [ZHS+03a], der die Skalierbarkeit von fehlertoleranten Overlay-Netzen verbessert. Diese vorgeschlagene Architektur baut auf einem strukturierten P2P-Overlay-Netz auf. Somit ist die Overlay-Architektur nicht vollvermascht wie bei RON, sondern hierarchisch untergliedert. Dies reduziert den nötigen Signalisierungsaufwand und ermöglicht eine bessere Skalierbarkeit. Der Datentransport erfolgt ähnlich wie bei RON. In den Endsystemen sind Erweiterungen nötig, die für eine Anbindung an das Overlay-Netz sorgen. Auch in diesem Vorschlag erfolgt der Datentransport in intransparenter Weise durch zusätzliche Header in Tunnelmechanismen. Ähnlich wie im Falle von RON werden die geschützten Datenpakete, auch für den Gutfall, ausschließlich innerhalb des Overlay-Netzes transportiert. Somit ist dieser Vorschlag für den Fall von seltenen Ausfällen ebenfalls nicht effizient.
Gegenüber RON ermöglicht dieser strukturierte Ansatz die Teilnahme von einer Vielzahl an Knoten im Overlay-Netz. Das zugrundeliegende strukturierte P2P-Netz, Tapestry [ZHS+03b], impliziert aber, dass Datenpakete auf dem Weg durch das Overlay-Netz

eine Vielzahl an Overlay-Knoten passieren müssen. Eine inhärente Eigenschaft von Tapestry ist, dass, je mehr Knoten am Overlay-Netz beteiligt sind, desto mehr Overlay-Knoten im Mittel von gerouteten Paketen durchlaufen werden müssen. Dies steht im Widerspruch zu veröffentlichten Messungen innerhalb des RON: In 97.8% der gemessenen Fälle bringt der Ansatz von RON, im Falle eines Ausfalls, nur einen einzigen weiteren Overlay-Knoten zu benutzen, klare Vorteile gegenüber der Benutzung von mehreren zusätzlichen Knoten im Overlay-Netz, wie dies Zhao et al. vorschlagen. Der Vorschlag von Zhao ist ebenfalls in der Skalierbarkeit begrenzt. Der Grund liegt unserer Ansicht nach daran, dass sowohl Signalisierung als auch der eigentliche Datentransport innerhalb des strukturierten P2P-Netzes erfolgt.

In der vorliegenden Arbeit schlagen wir einen Ansatz vor, der ebenfalls auf strukturierten P2P- bzw. Overlay-Netzen basiert ist. Wir benutzen diese Overlay-Netze aber ausschließlich zu Signalisierungs- bzw. Konfigurationszwecken. Der eigentliche Datentransport erfolgt dann außerhalb der Signalisierung in einem dedizierten minimalen Overlay-Netz. Datenpakete werden, angelehnt an RON, nur durch einen zusätzlichen Hop innerhalb des Overlay-Netzes geroutet.
Des Weiteren berücksichtigen die präsentierten Ansätze für fehlertolerante Overlay-Netze keine Gruppenkommunikation. Die Architekturen gehen von zu schützenden Unicast-Verbindungen zwischen kommunizierenden Endsystemen aus. Der Vorschlag dieser Arbeit deckt Anwendungen der Gruppenkommunikation ebenfalls ab.

3.3.2 Contentbasierte Overlay-Netze

Die in Abschnitt 3.3.1 präsentierten Vorschläge fehlertoleranter Overlay-Netze berücksichtigen die Systemfunktionalität* der zu schützenden Applikationen nicht. Im Wesentlichen konzentrieren sich diese Ansätze auf den reinen fehlertoleranten Transport von Daten. Einige der Ansätze schlagen zwar durch die Einführung von flow-spezifischen QoS-Parametern, eine einfache Kategorisierung des Datenverkehrs vor. Dies erlaubt aber nur eine einfache, allgemeine Berücksichtigung der Funktionalität von Applikation. Die Systemfunktionalität der Applikation wird dabei nicht berücksichtigt.
In diesem Abschnitt beschreiben wir Overlay-Netzarchitekturen, die explizit die Funktionalität von Applikationen unterstützen, und somit die Fehlertoleranz vom reinen Ausfall des Datentransports auf die Systemfunktionalität der zu schützenden Applikationen abstrahieren.

Einen Vertreter dieser Kategorie stellen die *Content Distribution-Netze* (CDN) dar. Einige in den letzten Jahren entstandene Ansätze haben mittlerweile das Forschungs- bzw. Entwicklungsstadium verlassen und werden schon vermarktet. Vertreter hierfür sind z.B. die *Edgesuite* der Firma Akamai [aka04], bzw. das CDN *Speedsuite* von Speedera [spe04].
Die Idee eines CDN ist die Verbesserung der User-Experience bei der Anfrage von Content

*Mit Systemfunktionalität der Applikationen meinen wir in diesem Zusammenhang die *eigentliche* Aufgabe der Applikation. Dies kann z.B. sein, einen bestimmten Film darzustellen. Von welcher Stelle des Netzes der zu übertragende Film kommt, ist dabei in dieser Sichtweise von untergeordneter Bedeutung. Der Film kann u.U. von verschiedenen Orten im Netz bezogen werden.

(z.B. Webseiten) von einem Server aus dem Internet. Dadurch, dass die Anfrage des Benutzers an einen Server mit repliziertem Content (Replika-Server) weitergeleitet wird, der bessere QoS-Eigenschaften für den anfragenden Client liefern soll, steht der angefragte Content einer Webseite in der Regel schneller zur Verfügung.
Ein Nebeneffekt ist natürlich auch die Skalierbarkeit. Durch die Nutzung verteilter Replika-Server können mehr Benutzeranfragen gleichzeitig bedient werden als durch einen zentralen Server.
In Abbildung 3.2 sind die wesentlichen Komponenten eines CDN dargestellt (siehe [Pen03]). Die Grundidee eines CDN ist, Daten vom ursprünglichen (originalen) Server auf Replika-

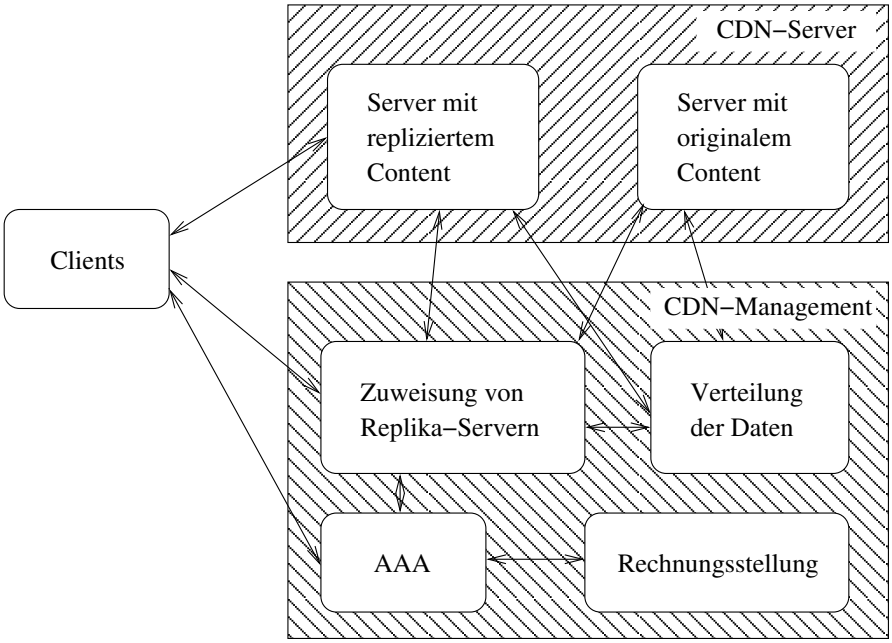

Abbildung 3.2: Komponenten eines Content Distribution-Netzes

Server im Netz zu kopieren. Clients, die anschließend entsprechende Daten anfragen, werden vom CDN zu einem Replika-Server umgeleitet, auf dem eine Kopie der Originaldaten zu finden ist. Da CDN schon von einigen Providern betrieben werden (es werden z.B. auch prominente Webseiten wie www.yahoo.com in einem CDN gehostet), ist u.U. auch ein Abrechnungssystem notwendig, das die Benutzer authentifiziert und deren Zugang regelt (AAA).

Eine ähnliche Vorgehensweise wird auch in verteilten *Cacheing-Systemen* [RSB01] angewandt. Wesentliche funktionelle Unterschiede zwischen CDN und verteilten Cacheing-Systemen bestehen in den folgenden Punkten:

- In CDN wird bestimmter Content bereits vor der ersten Anfrage aktiv repliziert.
- Bei Änderungen des originalen Content werden die replizierten Server ebenfalls aktiv vom CDN auf den aktuellen Stand gebracht.
- Das CDN weist bei Anfragen von Clients den Replika-Server selbst zu.

In Cacheing-Systemen erfolgt die Replizierung von Content erst nach der ersten Anfrage. Außerdem wird Content nicht aktiv von den Caches auf dem aktuellen Stand gehalten, sondern erst nach Ablauf von Timern u.U. erneut abgefragt.

Die vorgestellten, bereits in Betrieb befindlichen CDN der ersten Generation basieren auf der Annahme, dass die Verteilung der anfragenden Clients und die Popularität bestimmter Daten a priori bekannt ist.
Darauf basierend wurden *statische CDN* entwickelt. Mit *statisch* meinen wir hier die starre Festlegung von Replika-Servern für bestimmten Content, gemäß der angenommen Verteilung von anfragenden Clients im Netz.
Die Zuweisung der Replika-Server erfolgt dann durch Umleitungsmechanismen bei Anfragen von Clients. Möglich sind z.B. Umlenkungen durch das DNS (siehe Abschnitt 2.2), oder auch mittels HTTP. Die tatsächliche Situation eines bestimmten Replika-Servers (momentane Auslastung, Ausfall) wird von statischen CDN außer Acht gelassen. Der Fall, dass in einem CDN ein Replika-Server überlastet ist, gleichzeitig aber andere Replika-Server eines CDN eine geringe Auslastung haben, kann also durchaus vorkommen.
An diesem Punkt setzen Vorschläge zu dynamischen, selbstorganisierenden CDN an. SCAN [CKK02] ist ein Vertreter der dynamischen CDN. Fokus der Architektur ist vor allem die dynamische Platzierung von Content auf Replika-Servern, abgestimmt auf die Auslastung der Replika-Server und die Popularität des Content. SCAN basiert auf einem strukturierten P2P-Netz. Der Ansatz, grundlegende Ideen strukturierter, symmetrischer P2P-Netze, wie z.B. CAN [RFH+01] oder Tapestry [ZHS+03b], in CDN zu integrieren, die durch asymmetrische Client-Server Kommunikation gekennzeichnet sind, scheint dabei vielversprechend.
Das Projekt OceanStore [RWE+01] ähnelt einem dynamischen CDN in weiten Teilen. Dieses Overlay-Netz platziert Content ebenfalls dynamisch auf mehreren Replika-Servern. Der Fokus des Projektes ist jedoch die fehlertolerante zur Verfügungstellung von verteiltem Speicherplatz für dedizierte Benutzer. *Content* ist in diesem speziellen dynamischen CDN also nicht für alle Benutzer zugänglich, sondern nur für bestimmte, authentifizierte Benutzer.

Die Relevanz von CDN bzw. verteilten Cacheing-Systemen für diese Arbeit liegt im Grundprinzip der Fehlertoleranz durch redundanten Content. Kommt es zu einem Ausfall bei der Übertragung von Daten von einem Server zu einem anfragenden Client, dann sind Kopien der entsprechenden Daten u.U. noch auf einem anderen Replika-Server bzw. Cache vorhanden. Somit können derartige Architekturen die Systemfunktionalität der Applikation direkt unterstützen.
In den folgenden Kapiteln beschreiben wir einen schichtenübergreifenden Fehlertoleranzdienst, basierend auf Overlay-Netzen der auf der Transportschichtebene arbeitet und zusätzlich auch die Applikation bzw. den Content berücksichtigt.

3.4 Datentransport in Overlay-Netzen

3.4.1 Überblick über Verfahren

Ein gemeinsamer Nenner von fehlertoleranten Overlay-Netzen, die in das Routing von Datenpaketen eingreifen, kann im Ablauf des Datentransports innerhalb des Overlay-Netzes

gefunden werden. Laut unserem Kenntnisstand führen alle bis dato veröffentlichten Architekturen einen *zusätzlichen Header* dafür ein. Basierend auf diesem Header ist dann das Routing bzw. Ersatzschalten im Overlay-Netz organisiert.
Grundsätzlich kann zwischen zwei verschiedenen Vorgehensweisen unterschieden werden:

- Standardisierte Header
- Proprietäre Header

Erste Ansätze, wie das in Abschnitt 3.3.1 präsentierte DETOUR-Overlay-Netz [SAA+99], transportieren die Daten im Overlay-Netz durch IP-in-IP-Enkapsulierung [Sim95]. IP-in-IP-Enkapsulierung wurde ursprünglich entwickelt, um virtuelle Netze im Internet aufzusetzen. So basiert der Datentransport vieler Architekturen der frühen Overlay-Netze ebenfalls auf IP-in-IP-Enkapsulierung, wie in Abschnitt 2.2.1 angeführt.
Auch das Active Protocol Label Switching (APLS) [LJ03] führt standardisierte, universelle Header ein, die in einer Vielzahl unterschiedlicher Overlay-Netze verwendet werden können. Im Unterschied zur IP-in-IP-Enkapsulierung ist man mit APLS etwas flexibler. So können z.B. Pfade für ein Label Switching im Overlay-Netz flow-basiert geschaltet werden, was bei IP-in-IP nicht ohne Ergänzung möglich ist.

Einige fehlertolerante Overlay-Netze (siehe Abschnitt 3.3) führen proprietäre Mechanismen bzw. Header für den Datentransport ein.
In der RON-Architektur [ABKM01] werden ankommende Datenpakete zuerst mit einem speziellen *RON-Header* versehen. Anschließend wird das Datenpaket samt RON-Header über UDP/IP im RON weitergeleitet.
Eine Vielzahl an Overlay-Netzen baut auf dem strukturierten P2P-Netz *Tapestry* [ZHS+03b] auf. Auch in Tapestry wird, ähnlich wie bei RON, für den Datentransport ein zusätzlicher proprietärer Header eingeführt.

3.4.2 Einschränkungen für Ersatzschaltverfahren in Overlay-Netzen

Die bekannten Mechanismen des Datentransports in Overlay-Netzen (siehe Abschnitt 3.4.1) haben gewisse Nachteile aus der Sicht von möglichen Fehlertoleranzmechanismen. In den vorgeschlagenen Verfahren werden alle (originalen) IP-Datenpakete, die in ein Overlay-Netz aufgenommen, und anschließend innerhalb des Overlay-Netzes transportiert werden, durch zusätzliche Header vergrößert. Im Falle der IP-in-IP-Enkapsulierung kommt z.B. zusätzlich zum originalen IP-Paket noch ein weiterer IP-Header hinzu. In anderen Verfahren (z.B. RON) sind es proprietäre Header.
In diesem Kontext sind zwei Fälle von besonderem Interesse:

- **Lange Original-IP-Pakete**: z.B. Datenpakete mit, durch die MTU (Maximum Transfer Unit) festgelegter, maximal erlaubter Länge. Im Falle von Ethernet ist die MTU typischerweise durch eine Länge von 1500 Byte begrenzt.
- **Kurze Original-IP-Pakete**: z.B. TCP/IP Acknowledgements ohne Payload. Die übliche Paketlänge dieser IP-Pakete beträgt 52 Byte.

Werden *lange Original-IP-Pakete* in ein Overlay-Netz aufgenommen, führen die durch den Datentransportmechanismus notwendigen zusätzlichen Header dazu, dass die MTU überschritten werden würde. Das entstehende enkapsulierte IP-Paket kann dann im Internet nicht weiter transportiert werden. Es gibt dann prinzipiell zwei Möglichkeiten:

Das neue IP-Paket könnte fragmentiert werden, d.h. es wird in zwei eigenständige IP-Pakete unterteilt, die jeweils nicht größer sind als die zulässige MTU. Im Anschluss daran wäre es möglich diese zwei Fragmente zum nächsten Hop im Overlay-Netz zu transportieren. Dort würden sie wieder zu einem IP-Paket zusammengesetzt, Änderungen am Header vorgenommen und weiterverschickt werden. Ist ein weiterer Hop innerhalb das Overlay-Netzes das nächste Ziel, müsste wieder fragmentiert werden. Verlässt das IP-Paket das Overlay-Netz, kann das IP-Paket dekapsuliert werden, und man erhält wieder das Originale-IP-Paket.

Der beschriebene Mechanismus mit Fragmentierung ist wenig praktikabel, fehleranfällig und benötigt zusätzlichen Rechenaufwand. Aus diesem Grund kommt Fragmentierung in Overlay-Netzen praktisch nicht vor. Alle beschriebenen Architekturen weisen die Endsysteme an, Originale-IP-Pakete in das Overlay-Netz zu versenden die unter der zulässigen MTU liegen. Durch diesen Mechanismus ist gewährleistet, dass die im Overlay-Netz enkapsulierten IP-Pakete nicht größer sind als die zulässige MTU.
Diese nötige Reduzierung der Paketlänge auf Seiten der Endsysteme hat Auswirkungen auf die möglichen Fehlertoleranzmechanismen. So ist es nicht möglich die beschriebenen fehlertoleranten Overlay-Netze im Netz transparent, gegenüber den Endsystemen bzw. Benutzern, zu installieren. Reduziert das Endsystem die Paketlänge nicht, oder kann es sie nicht reduzieren, ist kein Datentransport in den vorgeschlagenen Overlay-Netzen möglich.

Werden *kurze Original-IP-Pakete*, wie z.B. TCP Acknowledgements, in das Overlay-Netz aufgenommen, ist Fragmentierung bzw. Transparenz oder Intransparenz der Architektur kein Diskussionspunkt. Durch die zusätzlich nötigen Header wird das Originalpaket vergrößert, bleibt aber trotzdem weit unterhalb der zulässigen MTU.
Betrachtet man allerdings für diesen Fall das Verhältnis der Größe des Original-IP-Pakets zu der Größe des enkapsulierten IP-Pakets, fällt für alle vorgeschlagenen Architekturen ein relativ großer Overhead auf. Ein reguläres Original-TCP/IP Acknowledgement hat die Größe von 52 Byte. Wird ein derartiges Paket z.B. in RON transportiert, dann kommen ein äußerer IP-Header, ein äußerer UDP-Header und der proprietäre RON-Header dazu. Insgesamt würde das Original-IP-Paket somit um 46 Byte verlängert werden, zu einer Gesamtgröße von 98 Byte. Diese beinahe Verdoppelung der Paketgröße ist nicht effizient und lässt Raum für Verbesserungen offen. Auch die anderen präsentierten Verfahren des Datentransports weisen ähnliche Ineffizienzen auf.

Im Rahmen dieser Arbeit präsentieren wir ein neues Verfahren des Datentransports in Overlay-Netzen, das die beiden beschriebenen Probleme umgeht. Das präsentierte Konzept der virtuellen Label (siehe Abschnitt 5.5) macht die Verlängerung von IP-Paketen bei der Enkapsulierung unnötig.

3.5 Fehlertoleranz in den Endsystemen

In den bisherigen Abschnitten wurden rein netzbasierte Fehlertoleranzmechanismen angeführt. Die Endsysteme bzw. Applikationen sind nur lose in diese Mechanismen integriert. Abgesehen von dem Anschluss an die Mechanismen, ist auf der Endgeräteseite keine weitere Aktion nötig bzw. möglich gewesen.
In diesem Abschnitt gehen wir auf Fehlertoleranzmechanismen ein, die in Interaktion mit den Endsystemen stehen bzw. von den Endsystemen ausgehen. Zunächst betrachten wir *protokollbasierte Fehlertoleranz* beim Datentransport. Des Weiteren präsentieren wir *contentbasierte Fehlertoleranzmechanismen*, die von den Endsystemen initiiert bzw. gesteuert werden.

3.5.1 Protokollbasierte Verfahren

Das Transmission Control-Protokoll (TCP) [Ins81] ist das grundlegende Protokoll für die zuverlässige Datenübertragung im Internet. Aus Sicht der Fehlertoleranz organisiert TCP die fehlerfreie Übertragung der Daten. Gehen Datenpakete im Netz verloren, stellt TCP dies nach einer gewissen Zeit fest und organisiert die erneute Sendung der verlorengegangenen Daten.
Die Applikationen werden dabei nicht berücksichtigt. TCP sorgt nur für den fehlerfreien Transport. Das in den letzten Jahren in vielen Details verbesserte TCP kann für Multimedia-Anwendungen, die keine allzu hohe Interaktivität benötigen, wie z.B. Video on Demand (VOD) [KLW01], gut eingesetzt werden. Die Applikationen wissen um, durch TCP induzierte, möglichen Verzögerungen, und können dieses Problem durch lokale Puffermechanismen umgehen.
TCP wurde auch entwickelt, um Fehler im Netz zu maskieren die statistischer Art sind. D.h. im Netz liegt kein Versagen im eigentlichen Sinn vor, durch Überlastsituationen treten aber statistische Verluste von Paketen auf. Derartige Fehler können in den Endsystemen somit relativ gut behandelt werden. Treten aber schwerwiegendere Fehler auf (siehe Abschnitt 3.2.2), die z.B. ein BGP-Rerouting und somit Konnektivitätsverlust zur Folge haben, dann kann auch TCP keine Abhilfe schaffen. Eine Verbindung würde in diesem Fall nach Ablauf eines Timers abbrechen, bzw. eventuell vorhandene lokale Puffer würden sich leeren und die betreffende Applikation müsste versagen.
Für TCP wurden in den letzten Jahren einige Erweiterungen vorgeschlagen. So gibt es Ansätze, die die Migration von TCP-Verbindungen von einem Endgerät zum anderen erlauben: z.B. *Server fault-tolerant TCP* (ST-TCP) [MMF03] oder *Fault Tolerant TCP* (FT-TCP) [ABEK+01]. Diese Ansätze maskieren Ausfälle von Servern vollkommen transparent auf der Transportschichtebene gegenüber einem Client. Auf Client-Seite ist dabei keine Änderung des Standard TCP/IP Stacks notwendig. Die Funktionalität ist ausschließlich auf Seite des Servers alloziert. Das *Packet Path Diversity TCP* (PPD-TCP) [SLG02] schlägt ebenfalls ein fehlertolerantes TCP vor. Basierend auf *vorausgesetzter* Verfügbarkeit von disjunkten Pfaden im Netz, wird ein Verfahren vorgeschlagen, das TCP-Acknowledgements redundant überträgt. Durch eine Erweiterung kann der redundante Pfad ebenfalls für den Datentransport benutzt werden.

Das Stream Control Transmission-Protokoll (SCTP) [OY02] baut nicht auf TCP auf, sondern ist ein eigenständiges Protokoll der Transportschicht. Während TCP-Verbindungen explizit an jeweils eine IP-Adresse der Kommunikationspartner gebunden sind, unterstützt SCTP mehrere IP-Adressen eines Kommunikationspartners. So ist es möglich, ein effizienteres Multi-Homing (siehe Abschnitt 3.2.2) zu realisieren. Multi-Homing mittels SCTP benötigt keinen Eingriff seitens des Netzes. Die BGP Forwarding-Tabellen müssen nicht aktualisiert werden. Somit ist ein Umschaltvorgang im SCTP basiertem Multi-Homing nicht von Konvergenzzeiten abhängig, und damit schnell realisierbar. Das erst kürzlich entwickelte SCTP eignet sich somit gut, um Ausfälle in der Anbindung an das Internet zu maskieren (z.B. Ausfall einer Netzwerkkarte in den Endsystemen bzw. Ausfall im lokalen LAN). Da SCTP aber keinen, bzw. nur stark limitierten, Einfluss auf die Routen im Internet zwischen den Kommunikationspartnern nehmen kann, ist es bei Ausfällen innerhalb des Netzes kein Ersatz für die präsentierten Ansätze fehlertoleranter Overlay-Netze.

Der in den folgenden Kapiteln präsentierte netzbasierte Fehlertoleranzdienst ergänzt sich mit vorgestellten Ansätzen der Fehlertoleranz durch Protokolle in den Endsystemen. Für die Arbeit von Relevanz ist die Verknüpfung im Einzelfall. So muss der präsentierte Fehlertoleranzdienst z.B. mit Multi-Homing durch SCTP kooperieren können. In den relativ starren Vorschlägen, präsentiert in Abschnitt 3.3.1, ist dies nicht vorgesehen.
Da auf dem Gebiet der fehlertoleranten Protokolle weiterführende Vorschläge zu erwarten sind, ist der Ansatz dieser Arbeit von Vorteil: Durch die programmierbare Architektur des Fehlertoleranzdienstes können neue Ansätze auf dem Gebiet der fehlertoleranten Protokolle flexibel integriert werden.

3.5.2 Anwendungsbasierte Verfahren

Die in Abschnitt 3.5.1 präsentierten Fehlertoleranzmechanismen in Endsystemen berücksichtigen den Content bzw. die Funktionalität von Anwendungen nicht, oder nur am Rande. Der Fokus liegt klar auf der fehlertoleranten Ende-zu-Ende-Übertragung von Daten im Internet.
In diesem Abschnitt betrachten wir Fehlertoleranzmechanismen, die in den Applikationen auf den Endsystemen bzw. im Content direkt verankert sind. Durch eine angepasste, spezifische Kodierung des Content kann der Content selbst gewisse Verluste tolerieren.
Eine Möglichkeit sind sog. *Erasure Codes*, auch *Forward Error Correction*-Kodierung genannt. Grundprinzip dieser Art von Kodierung ist, einen Originaldatenstrom, der aus N Datenpaketen besteht, in einen Datenstrom mit M Paketen zu kodieren. Dabei gilt: $M > N$. Diese M Pakete werden anschließend über das Netz verschickt. Ein Sender muss von diesen M Paketen nur eine Mindestanzahl K Pakete empfangen ($K < M$). Aus den empfangenen K Paketen können die N Datenpakete des Originaldatenstroms bzw. der originale Content auf Empfängerseite berechnet werden. Der Ansatz *Digital Fountain* [BLM02] schlägt vor, dieses Prinzip zu nutzen, um große Datenmengen in einem Netz zuverlässig zu verteilen. Die Autoren sehen die Vorteile dieses Vorschlags in der möglichen Verbesserung der Skalierbarkeit von *Reliable Multicast*-Anwendungen.
Die beschriebenen Erasure Codes fügen eine zusätzliche Wartezeit in den Datentransport mit ein. So muss der Empfänger warten, bis er die nötige Mindestanzahl an Datenpaketen empfangen hat, um den ursprünglichen Content berechnen zu können. Diese zusätzliche

Wartezeit schränkt die möglichen Anwendungen ein. Prinzipiell können alle Daten mittels *Erasure Codes* geschützt werden, wenn für sie gilt, dass sie auf Empfängerseite erst weiterverarbeitet werden, wenn sie vollständig übertragen worden sind. Dies ist z.B. für Softwaredownloads der Fall.
Für Multimedia Streaming-Anwendungen hingegen ist das in Digital Fountain angewandte Kodierungsprinzip ungeeignet. Ein Empfänger müsste warten, bis er die vollständigen Daten erhalten hat. Danach könnte erst mit dem Abspielen begonnen werden.
Unabhängig davon hat die Kodierung von Multimedia-Datenströmen ein anderes Ziel. Während Erasure Codes Redundanz hinzufügen, und damit die Fehlertoleranz bei der Übertragung erhöhen, hat die Kodierung von Video-Datenströmen in erster Linie das Ziel, die zu übertragenden Daten zu komprimieren. Dabei werden im Originaldatenstrom vorhandene Redundanzen minimiert. In MPEG 2 (siehe [fS94]) geschieht dies z.B. durch die Einführung von drei unterschiedlichen Arten von Frames (Bildern). *Intra-Frames* (I-Frames) stellen Bilder dar, die unabhängig von Frames sind, die zeitlich vor oder nach den I-Frames dargestellt werden sollen. Die zwei anderen Arten von Frames (B und P) sind abhängig von Bildern, die zeitlich vor oder nach dem jeweiligen Frame dargestellt sind. Durch diese Abhängigkeit ist es möglich, Redundanzen zu eliminieren. Es werden im Wesentlichen nur Differenzen zwischen Bildern übertragen. Geht somit bei der Übertragung im Netz ein Datenpaket verloren, das Daten enthält, wovon zukünftige oder bereits empfangene Daten abhängig sind, können diese abhängigen Daten für die Darstellung eines Videostroms ebenfalls nicht verwendet werden. Aus der Sicht der Fehlertoleranz ergibt sich somit ein wichtiger Aspekt: Datenpakete eines Datenstroms können unterschiedlich priorisiert werden. Geht z.B. ein einzelnes Datenpaket eines I-Frame verloren, hat dies im darzustellenden Video u.U. einen mehrere Sekunden dauernden Ausfall zur Folge. Fällt hingegen nur ein Datenpaket aus, das einen P-Frame enthält, dann wird dies u.U. vom Benutzer gar nicht bemerkt.
Mögliche Fehlertoleranzmechanismen in diesem Zusammenhang sind z.B. dynamisches Einfügen von zusätzlichen I-Frames auf der Senderseite [Hei01]. Auch Verfahren, die Datenpakete gemäß ihren Auswirkungen auf die Darstellung im Falle eines Verlustes priorisieren und entsprechend schützen, sind aufbauend auf der grundlegenden Arbeit von Albanese et al. *Priority Encoded Transmission* [ABE+94] denkbar.
Liang et al. schlagen vor, einen Originaldatenstrom in mehrere Teildatenströme zu kodieren, die über unkorrelierte, disjunkte Pfade im Netz von einem Sender zu einem Empfänger geschickt werden [LSG01]. Die Teildatenströme werden über *Multiple Description Coding* [JO00] erzeugt. Der Beitrag von Liang fokussiert sich auf die Applikationsebene und präsentiert mögliche Vorteile dieses Fehlertoleranzmechanismus. Jene Arbeit geht von durch niedere Netzschichten zur Verfügung gestellten, unkorrelierten und disjunkten Pfaden durch das Internet aus.
Eine weitere Möglichkeit für Applikationen bei auftretenden Übertragungsfehlern einzugreifen ist die serverseitige Kodierung von Datenströmen in *variable Bit-Raten* (VBR) [LOR98]. Dabei sind Datenströme des gleichen Typs (Films) in unterschiedlicher Qualität auf einem Server vorhanden. Die Qualitätsunterschied beeinflusst direkt die notwendige Übertragungskapazität im Netz. Stellt ein Client statistische Paketverluste bei einer Datenübertragung fest, kann er auf eine niedere Qualität und damit geringere Übertragungsrate wechseln.
Während die Optimierung an lokale Gegebenheiten, im beschriebenen Fall von VBR, aus-

schließlich von den beteiligten Endsystemen geregelt wird, beschreiben Methoden der Transkodierung einen anderen Weg. Steht der gewünschte Datenstrom z.B. nicht in VBR-Kodierung zur Verfügung, gibt es Ansätze, die zur Unterstützung der Empfänger innerhalb des Netzes eine dynamische Qualitätsreduktion und damit QoS-Anpassungen vornehmen [KCD+00]. Dies kann seitens der Empfänger initiiert werden.

Die in den Abschnitten 3.5.1 und 3.5.2 präsentierten Ansätze der Fehlertoleranz in den Endsystemen wurden in erster Linie entwickelt, um statistische Paketverluste bei der Übertragung von Daten zu behandeln. Diese vorgeschlagenen Verfahren können die tatsächliche Situation im Netz aber nicht berücksichtigen, da sie zum einen ausschließlich die Applikationsschicht betrachten, und zum anderen nur auf den beteiligten Kommunikationspartnern alloziert sind. So sind protokoll- als auch anwendungsbasierte Verfahren in Endsystemen, z.B. für den Fall eines BGP-Rerouting und dem damit u.U. verbundenen Verlust der Konnektivität, machtlos.
Ein weiteres Defizit der beschriebenen Mechanismen liegt in der mangelnden schichtenübergreifenden Zusammenarbeit der Fehlertoleranzmechanismen. Die meisten beschriebenen Verfahren laufen für sich isoliert ab. Jüngste Ansätze, wie das beschriebene fehlertolerante PPD-TCP, streben jedoch nach mehr Interaktion mit der darunter liegenden Middleware bzw. auch mit der Anwendungsebene. So setzt PPD-TCP allgemein die Interaktion mit einem fehlertoleranten Overlay-Netz voraus, ohne dies weiter zu spezifizieren. Die in Abschnitt 3.3 präsentierten Ansätze erscheinen dafür aber aufgrund mangelnder Transparenz und Skalierbarkeit wenig geeignet.

3.6 Vergleich der bestehenden Ansätze für Fehlertoleranz

In diesem Abschnitt stellen wir die präsentierten Fehlertoleranzmechanismen der internetbasierten Kommunikation gegenüber. Zunächst führen wir Kategorien zur Klassifizierung ein. Demgemäß ordnen wir die beschriebenen Mechanismen ein und grenzen den Beitrag dieser Arbeit ab.

3.6.1 Kategorien zur Klassifizierung

Ziel dieses Abschnitts ist die Einführung von Kategorien, die zur Unterscheidung der präsentierten Fehlertoleranzmechanismen herangezogen werden können.
Im Rahmen dieser Arbeit unterscheiden wir Fehlertoleranzmechanismen anhand von vier übergeordneten Merkmalen:

- Ort der Implementierung
- Möglichkeiten der Fehlermaskierung bzw. Fehlerabschwächung
- Die Arten von behandelbaren Fehlern
- Adaptierbarkeit an Applikationen

Der *Ort der Implementierung* gibt an, wo und auf welcher Ebene der Mechanismus im Netz platziert ist. Im Detail kann man unterscheiden nach topologischer Platzierung und nach der Platzierung im Schichtenmodell.
Die *Möglichkeiten der Fehlermaskierung* charakterisieren den Fehlertoleranzmechanismus: Gemäß der in Abschnitt 4.2.2 eingeführten Arten der Fehlermaskierung unterscheiden wir in horizontale, vertikale und longitudinale Maskierung.
Grundlage für diese Arbeit ist die Maskierung bzw. Abschwächung (engl. mitigation) von Fehlern, die auf Fehlerzustände des Netzes bei der Datenübertragung zurückzuführen sind. Als Indikator für Fehlerzustände im Netz dienen am Netzrand gemessene Paketverluste. Aus diesem Grund unterscheiden wir in *statistische Paketverluste* aufgrund von Überlast, sowie in länger *andauernde Paketverluste* aufgrund von Versagen innerhalb des Internets.
Auch die *Adaptierbarkeit* der Fehlertoleranzmechanismen an Applikationen ziehen wir für die Klassifizierung heran. Die direkte Unterstützung von Applikationen erfordert ein detailliertes Wissen der Fehlertoleranzmechanismen über die Funktionalität der Applikationen. Somit ist es erforderlich, Fehlertoleranzmechanismen an Applikationen anzupassen.

3.6.2 Klassifizierung von bestehenden Fehlertoleranzmechanismen

In Abbildung 3.3 sind die vorgestellten Fehlertoleranzmechanismen nach Merkmalen gegenübergestellt. Die in den Netzarchitekturen bzw. in der Internetschicht verankerten Me-

FT–Architekturen/ Mechanismen/ Dienste	Kategorien	Topologisches Wirken: Innerhalb technologischer oder administrativer Domänen	Topologisches Wirken: Domänenübergreifend/ heterogen	Topologisches Wirken: im Netz	Topologisches Wirken: am Edge	Topologisches Wirken: flexibel platzierbar	Eingriff in Layer: Layer 2 basiert	Eingriff in Layer: Layer 3/4 basiert	Eingriff in Layer: Applikationsschicht basiert	Fehlermaskierung bzw. Mitigation: Vertikal Maskierung: Applikations unabhängig	Fehlermaskierung bzw. Mitigation: Vertikale Maskierung/Mitigation: Adaptierung Content	Fehlermaskierung bzw. Mitigation: Horizontale Maskierung: Redundante Pfade	Fehlermaskierung bzw. Mitigation: Longitudinale Maskierung: Redundanter Content	Fehlermaskierung bzw. Mitigation: Schnelle Konvergenz	Fehlermaskierung bzw. Mitigation: Transparenz gegenüber Applikationen	Geeignet für Art von Fehler: Versagen im Netz	Geeignet für Art von Fehler: Versagen im Zugangsbereich	Geeignet für Art von Fehler: Ausfall Datentransport: kurzzeitig	Geeignet für Art von Fehler: Ausfall Datentransport: längerfristig	Geeignet für Art von Fehler: Überlast: statistische Paketverluste	Geeignet für Art von Fehler: Contentausfall	Adaptierbarkeit an Applikation: Schützen von Anwendungen der Gruppenkommunikation	Adaptierbarkeit an Applikation: Maskierung bzw. Mitigation: Anpassung an Applikationen	Adaptierbarkeit an Applikation: Direkte Unterstützung von Applikationen	Adaptierbarkeit an Applikation: Rapid Service Creation	Adaptierbarkeit an Applikation: geringer Aufwand/ Kosten bei Änderungen	Skalierbarkeit: Verbreitung	Skalierbarkeit: Anzahl Nutzer
Architekturen	FT nach 3.2.1 (ATM, MPLS, OTN)	X		X	X		X	(X)				X		X	X	X	X	X	X	(X)								X
	BGP		X	X				X				X			X	X		(X)	X								X	X
	FASRO		X	X				X				X			X	X		X	X								X	X
	Multi–Homing		X		X			X				X			X		X		X								X	X
Overlay–Netze	RON		X		X			X				X		X		X		X	X	(X)						(X)		(X)
	Zhao structured RON		X		X			X	X			X	X	X		X		X	X		(X)	(X)				(X)		(X)
	Statische/ Dynamische CDN	X	X		X				X				X	X	X	X			X		X	(X)						(X)
	Verteilte kooperierende Cachespeicher	X	X		X				X				X	X	X	X			X		X	(X)				(X)		(X)
Dienste/ Appl.	TCP, ST–TCP, FT–TCP Puffermechanismen	X	X		X				X	X			(X)	(X)	X	X		(X)		X	(X)			X			X	X
	PPD–TCP		X		X			X		X		X		X	X	X		X	X	X				X				X
	SCTP		X		X			X		X		(X)		X	X		X	(X)	X								X	X
	Digital Fountain	X	X		X				X		X			(X)		X		(X)		X				X			X	X
	Liang: MDC–basierte Pfad–Diversität		X		X			X	X		X	X		X		X		X	X	X								X
	Kodierung: VBR, Transkodierung	X	X		X	X			X		X			X	X	X				X		(X)	X				X	X
Vorschlag dieser Arbeit		(X)	X	(X)	X	X		X	X	X	X	X	X	X	X	X	(X)	X	X	(X)	X	X	X	X	X	X	X	X

Abbildung 3.3: Kategorisierung von Fehlertoleranzmechanismen

chanismen beschränken sich rein auf horizontale Maskierung. Das bedeutet, man nutzt alternative Pfade im Falle eines Ausfalls beim Datentransport. Es gibt domänenbegrenzte Ansätze, die Anforderungen unterschiedlicher Applikationen in Gruppen aggregieren und sie entsprechend unterschiedlich behandeln. Domänenübergreifend werden spezifische Anforderungen von Applikationen an einen fehlertoleranten Datentransport aber im Wesentlichen nicht berücksichtigt.
Fehlertolerante Overlay-Netze liefern sowohl horizontale als auch longitudinale Maskierung von Fehlern. Die präsentierten Ansätze haben im Vergleich zu den niederschichtigen, domänenübergreifenden Mechanismen bei einem Ausfall eine kürzere Konvergenzzeit. Eine transparente, horizontale Fehlermaskierung gegenüber Applikationen ist jedoch nicht möglich. Des Weiteren skalieren die beschriebenen Ansätze nicht: Die Anzahl der beteiligten Elemente der Overlay-Netze (Knoten) ist limitiert. Somit ist die Verbreitung der Ansätze begrenzt.
Die präsentierten Ansätze in den Endsystemen ermöglichen eine horizontale und vertikale Fehlermaskierung bzw. Mitigation. Die Horizontale Maskierung in den Endsystemen konzentriert sich im Wesentlichen auf den Zugangsbereich. Ansätze wie PPD-TCP, erweitern die horizontale Maskierung auch auf das Netz, setzen aber beschriebene Ansätze fehlertoleranter Overlay-Netze voraus, und haben somit, durch die jeweiligen Architekturen bedingt, Limitierungen.

Die präsentierten Mechanismen für domänenübergeifende, horizontale Fehlermaskierung sind entweder transparent mit langer Konvergenzzeit und skalieren, oder intransparent mit kurze Konvergenzzeit und skalieren nicht im allgemeinen Fall. Es fällt auf, dass kein Ansatz eine durchgängige Kombination aus horizontaler, vertikaler und longitudinaler Fehlermaskierung bzw. Mitigation liefert. Des Weiteren sind bestehende Fehlertoleranzmechanismen **nicht einfach** – im Sinne einer Rapid Service Creation – an neue Applikationen anpassbar.

Im Rahmen dieser Arbeit präsentieren wir einen Fehlertoleranzmechanismus, der sich aus drei kombinierbaren Modulen (horizontaler-, vertikaler- und longitudinaler Fehlermaskierung) zusammensetzt. Aufbauend auf einem neuen Transportmechanismus für Daten in einem Overlay-Netz ist Transparenz gegenüber Applikationen möglich. Durch die Nutzung der Technologie der Overlay-Netzes ist eine kurze Konvergenzzeit gegeben. Der Fehlertoleranzmechanismus skaliert durch die in dieser Arbeit vorgeschlagene Trennung von Signalisierung und Transportfunktionen.
Darüber hinaus ermöglicht die vorgeschlagene Bereitstellung der Fehlertoleranz mittels programmierbarer Netze die flexible Anpassung an Anforderungen von kommenden Applikationen.

3.7 Architekturen programmierbarer Knoten

Die Forschung auf dem Gebiet der *Programmierbaren Netze* führte in letzter Zeit zu einer Vielzahl von Vorschlägen für programmierbare Knoten bzw. Router. In diesem Abschnitt geben wir einen Überblick über die einzelnen Ansätze. Im Wesentlichen gehen die Vorschläge in zwei unterschiedliche Richtungen.
Zum einen gibt es Ansätze, die auf Standardsystemen aufbauen. So wird auf Standardhardware und Betriebssystemen aufgesetzt, um rein softwarebasiert die Basisfunktionalität

programmierbarer Knoten zur Verfügung zu stellen.
Zum anderen werden Systeme mit dedizierter Hardware vorgeschlagen, die in der entwickelten Software den Standardsystemen ähneln.

3.7.1 Auf Standardhardware basierte Systeme

Programmierbare Knoten, die auf Standardhardware aufbauen, wie z.B. ein leistungsfähiger PC oder eine Workstation, sind dadurch charakterisiert, dass ein Knoten nur aus einer Maschine bzw. einem Rechner besteht. Durch den Einbau mehrerer Netzwerkkarten können Knoten dieser Art als ein Gateway zum Internet in Local Area Networks eingesetzt werden. Auf der Standardhardware, dieser sog. *Single Machine-Architekturen*, läuft ebenfalls ein Standard-Betriebssystem (z.B. Linux, FreeBSD). Die verschiedenen Vorschläge (z.B. ANTS [WGT98], Switchware [AAH+98], Snow on Silk [SGPW02], PromethOS [KRGP02], AMnet 2 [FHSZ02]) modifizieren und erweitern Teile des Betriebssystems und implementieren die benötigte Funktionalität für einen programmierbaren Knoten somit rein in Software. Die in Abbildung 2.4 dargestellte Einbindung der Netzschichten macht Eingriffe in das Betriebssystem notwendig, da sonst keine durch den Knoten durchlaufende Datenpakete abgegriffen werden können. Prinzipiell kann die eigentliche Paketverarbeitung dann auf zwei unterschiedliche Weisen erfolgen:
Zum einen können Datenpakete im Kern des Betriebssystems abgegriffen und dort direkt verarbeitet werden, wie dies z.B. in PromethOS vorgesehen ist. Das nötige Knotenmanagement läuft ausserhalb des Betriebssystems, im sog. *Userspace*, als regulärer Prozess. Die Paketverarbeitung, d.h., der programmierbare Dienst, läuft als ein Modul im Kern ab. Für die Kommunikation zwischen Knotenmanagement und den paketverarbeitenden Modulen im Kern ist eine umfangreiche Schnittstelle notwendig. So müssen Module z.B. vom Management in den Kern geladen und überwacht werden.
Zum anderen besteht die Möglichkeit darin, Datenpakete, wie z.B. in *Snow on Silk*, im Kern abzugreifen und über eine Schnittstelle direkt an Prozesse ausserhalb des Betriebssystems zu übergeben. Die Paketverarbeitung läuft dann ausserhalb des Kerns ab. Nach abgeschlossener Verarbeitung werden die bearbeiteten Pakete wieder in den Kern kopiert und setzen ihren Weg fort. Im Kern des Betriebssystems ist somit nur eine Implementierung der Schnittstelle notwendig, die Datenpakete in den Userspace übergibt und von dort empfangen kann.
Es gibt auch Mischformen von Architekturen. Sie verarbeiten Datenpakete sowohl im Kern als auch im Userspace, wie dies z.B. in AMnet 2 vorgesehen ist.

Diese, in jüngster Zeit entwickelten, und teilweise durch Implementierung validierten, Knotenarchitekturen haben mittlerweile ein Niveau erreicht, dass sie in Forschungsnetzen zur Entwicklung von programmierbaren Diensten eingesetzt werden können. Ausserhalb von Testnetzen werden die beschriebenen Architekturen aber noch nicht eingesetzt. Nach unserer Ansicht liegt dies im Wesentlichen an drei offenen Punkten: *Skalierbarkeit*, *Sicherheit* und nicht zuletzt der *Zuverlässigkeit bzw. Fehlertoleranz*. Das offensichtliche Problem liegt sicher in der Skalierbarkeit der Architekturen. Sollen rechenintensive, programmierbare Dienste, wie z.B. die Transkodierung von Datenströmen [KCD+00] auf *Single Machine-Architekturen* bereitgestellt werden, stößt ein einzelner Rechner mit der Bereitstellung des Dienstes relativ schnell an Grenzen. Mit der gegenwärtigen Standardhardware ist es in Echtzeit schon

nahezu unmöglich z.B. nur einen einzigen H.264-Datenstrom zu dekodieren und wieder neu zu enkodieren, wie das eine fortgeschrittene Transkodierung erfordern würde. Auch die stetige Weiterentwicklung der Standardhardware, und damit verbesserte Performanz der zur Verfügung stehenden Hardware, kann hier keine Abhilfe schaffen. Es sollte ja nicht nur ein Datenstrom, sondern u.U. eine Vielzahl an Datenströmen parallel transkodiert werden. Der kontinuierlich steigende Bedarf an Übertragungsrate und Rechenleistung [GSF95] einiger Multimedia-Applikationen verstärkt das Problem zusätzlich.
Die vorgeschlagenen *Single Machine-Architekturen* sind jedoch gut geeignet für Dienste, die, im Vergleich zur Transkodierung, keine hohen Anforderungen an die zur Verfügung stehende Rechenleistung auf den programmierbaren Knoten stellen. Ein Beispiel hierfür wäre der *Active Multicast*-Dienst, realisiert in der Python-Architektur von Baumgartner [BBB02]. Dieser Dienst stellt ein programmierbares Application Layer Multicast Overlay-Netz (siehe 2.2.3) zur Verfügung.

3.7.2 Mehrrechnersysteme mit dedizierter Hardware

Um die Probleme der Skalierbarkeit bzw. Performanz von *Single Machine*-Architekturen umgehen zu können, wurden Mehrrechnersysteme (*Multi Machine*-Architekturen) eines programmierbaren Knoten, aufbauend auf teilweise dedizierter Hardware, entwickelt.
Decasper schlägt in [DPP99] eine *Active Node Architecture* (ANN) vor, die auf speziell entwickelter Hardware aufbaut. In diesem programmierbaren Knoten sind mehrere sog. *Active Network Processing-Elemente* (ANPE) intern über ATM miteinander verbunden. Diese stellen die Komponenten des programmierbaren Knoten dar. In den Router eingehende (IP-) Datenströme werden intern auf virtuelle ATM-Kanäle abgebildet. Diese werden dann durch den internen ATM-Switch an die ANPE vermittelt. In der Vermittlung wird mittels Lastbalancierung die Überlast einzelner ANPE verhindert. Die einzelnen ANPE im ANN ähneln der Architektur von *Single Machine-Architekturen*. Auf den ANPE läuft ein Standard-Betriebssystem. Die intern über ATM vermittelten IP-Pakete landen zunächst im Kern dieses Betriebssystems. Die weitere Paketverarbeitung ist über Kern-Userspace-Kommunikation identisch zu den in Abschnitt 3.7.1 beschriebenen Mechanismen. Nach der Verarbeitung verlässt ein Paket sein ANPE und wird, wieder intern mittels ATM, an den entsprechenden Ausgang des (IP-) Routers vermittelt.
Kuhns schlägt eine ähnliche Architektur [KDK+02] vor. Im Unterschied zu ANN ist es in dieser *Multi Machine*-Architektur eines programmierbaren Knotens möglich, in den einzelnen Rechnern (ANPE) die Pakete direkt im Kern des Betriebssystems zu verarbeiten.
Die Probleme der Skalierbarkeit bzw. Performanz der *Single Machine*-Architekturen können durch die präsentierten *Multi Machine*-Architekturen programmierbarer Router umgangen werden. Da aber die Architekturen der einzelnen Rechner (ANPE) in Mehrrechner-Systemen den Architekturen der *Single Machine*-Architekturen ähneln, bleiben die Probleme der Sicherheit und Zuverlässigkeit von programmierbaren Knoten nach unserer Ansicht weiter offen; Darauf gehen wir im nächsten Abschnitt ein.
Die in [Har02] angeführte Flexible High Performance-Plattform (FHIPPs) schlägt eine programmierbare Router-Architektur vor, die eine klassische, mikroprozessorbasierte Hardware mit Spezialprozessoren (DSP) und rekonfigurierbarer Hardware (FPGA) kombiniert. Jener Vorschlag erweitert Ansätze von *Single Machine-Architekturen* dadurch, dass zusätzliche

dedizierte Hardware in Standard-Workstation-basierte, programmierbare Router integriert wird. Dieser Ansatz ermöglicht es, Rechenaufgaben vom Mikroprozessor einer Workstation auf die (z.B. über einen PCI-Bus) angebundene, zusätzliche Hardware auszulagern. Die Arbeit baut darauf auf, dass gewisse Rechenoperationen in dedizierter Hardware wesentlich schneller ausgeführt werden können als in Software auf universellen Mikroprozessoren. Die Integration derartiger, dedizierter Hardware in Single Machine-Architekturen kann daher, abhängig von den zu verarbeitenden Daten, zu Leistungssteigerungen führen und ist somit eine gute Ergänzung für programmierbare Router.

3.7.3 Sicherheit und Zuverlässigkeit der Systeme

Das Grundprinzip der Programmierbaren Netze ist es, Dienste (repräsentiert durch Software-Code) auf einen programmierbaren Knoten im Netz zu laden und diesen dort auszuführen. Diese ausgeführten Dienste behandeln Datenströme, die den Router passieren.
Die präsentierten *Single Machine-* als auch *Multi Machine-*Architekturen laden Dienste in den Kern des Betriebssystems oder in den Userspace und führen sie aus. Wird ein Dienst in den Betriebssystemkern geladen, hat dieser vollen Zugriff auf das System und ist nur schwer zu kontrollieren. Die Palette möglicher Sicherheitsbedrohungen reicht z.B. von Programmierfehlern, die das System zum Stillstand bringen können, bis hin zu im Code versteckten Angriffen, die absichtlich zu einen nicht vorhersehbaren Systemverhalten führen können.
Architekturen, die Dienste nicht in den Kern laden, haben den Vorteil, dass ausgeführte Dienste im Userspace besser überwacht werden können, und somit gegebenenfalls auf Unregelmäßigkeiten reagiert werden kann. Die notwendige Schnittstelle derartiger Architekturen benötigt eine direkte Interaktion mit dem Kern, um Pakete zu transferieren. Dies ist dadurch realisiert, dass Dienste im Userspace mit umfangreichen Rechten ausgestattet sein müssen (z.B. root-Rechte), die ebenfalls wieder ähnliche Bedrohungen ermöglichen wie die rein kernbasierten Implementierungen. Im Gegensatz zu den kernbasierten Implementierungen können Userspace-basierte Dienste aber überwacht und reglementiert werden [HSS+02].
Somit ergeben sich einige Aspekte, die die Sicherheit von programmierbaren Systemen betreffen. Zusätzlich zu der teilweise schwierigen Überwachung von Diensten (z.B. benutzte Ressourcen) werden zur Gewährleistung der sicheren Ausführung von Diensten auf programmierbaren Knoten prinzipiell zwei unterschiedliche Kategorien von Vorschlägen gemacht:

1. Für programmierbare Dienste werden eigene Programmiersprachen entwickelt (z.B. [HK99, AMK+01]). Diese Sprachen limitieren die Möglichkeiten der Programmierung und begrenzen mögliche Dienste letztlich in der Funktionalität. Durch diese Limitierungen ist aber gleichzeitig sichergestellt, dass Dienste dem programmierbaren Knoten nicht schaden können. Einfache Limitierungen der Programmierung sind z.B. die Unterbindung von Schleifen. Somit können keine ungewollten (oder im Falle eines Angriffs gewollten) Endlosschleifen das programmierbare System blockieren. Auch weitere für die Sicherheit des Systems relevante Operationen, sind durch diese Programmiersprachen unterbunden.

2. Der Zugang zu programmierbaren Systemen wird reglementiert. Über Mechanismen wird festgelegt, dass nur bestimmte, bereits verifizierte Dienste geladen werden können.

Eine weitere Möglichkeit ist, dass nur authentifizierte Personen Dienste auf ein programmierbares System laden können. Auch Kombinationen sind denkbar. Da es nicht möglich ist, fehlerfreien Code automatisch zu verifizieren, bauen diese vorgeschlagenen Mechanismen letztlich auf dem *Prinzip des Vertrauens* auf [BACS02].

Beide Vorschläge schränken die Möglichkeiten programmierbarer Netze ein. Die Benutzung dedizierter Programmiersprachen begrenzt die Mächtigkeit möglicher Dienste stark, bzw. verhindert einige Dienste sogar. Komplexe Dienste, die z.B. eine Transkodierung von Datenströmen im Netz durchführen [KCD+00], benötigen für die Umsetzung im allgemeinen dedizierte Software-Bibliotheken. In den meisten Fällen dürfte es unmöglich bzw. mit erheblichem Aufwand verbunden sein, diese Bibliotheken in die vorgeschlagenen sicheren Sprachen zu migrieren.
Der Ansatz, nur einen vorher verifizierten Code bzw. Dienst auf ein programmierbares System laden zu können, zieht eine zeitliche Verzögerung zwischen Implementierung des Dienstes und der Ausführung nach sich. Dies beschränkt programmierbare Systeme in der möglichen Dynamik. Darüber hinaus kann auch durch umfangreiche Verifikation nicht ausgeschlossen werden, dass sich ein Dienst fehlerhaft verhalten könnte.

In der Literatur werden zwar fehlertolerante Applikationen mit Programmierbaren Netzen vorgeschlagen [HE00]. Die Zuverlässigkeit der zugrundeliegenden programmierbaren Plattformen wurde aber bisher vernachlässigt. Diesen Punkt behandeln wir in dieser Arbeit. Wir betrachten Sicherheitsaspekte der programmierbaren Knoten nur aus dem Blickwinkel der Fehlertoleranz. Der Zusammenhang liegt auf der Hand. Ein Versagen des programmierbaren Knoten, bedingt durch fehlerhafte Dienste, soll verhindert werden. Ob Dienste nun absichtlich oder unabsichtlich Fehlerzustände im System erzeugen, wird im Folgenden nicht weiter unterschieden.
Vorschläge, die auf Vertrauen aufbauen, können die Sicherheit eines programmierbaren Systems zwar verbessern, ein Restrisiko bleibt aber. Die entwickelten sicheren Programmiersprachen können die gewünschte Sicherheit zwar relativ gut gewährleisten, sind aber wenig praktikabel, da komplexere Dienste damit nicht realisiert werden können.
Einen Ausweg aus diesem Dilemma präsentieren wir im Rahmen dieser Arbeit. Im Gegensatz zu den präsentierten Ansätzen schlagen wir eine komponentenbasierte Architektur eines programmierbaren Systems vor. Darin sind die Management- und Steuerungsfunktionen eines Knotens, der Abgriff von Datenpaketen, und die Bereitstellung von Diensten räumlich getrennt.
Dadurch haben eventuell fehlerhafte Dienste keine Möglichkeit auf andere Komponenten einzuwirken, wie dies in den präsentierten Architekturen (siehe 3.7.1 und 3.7.2) der Fall ist. Außerdem können diese Komponenten redundant ausgelegt werden, was die Zuverlässigkeit weiter erhöht.

3.7.4 Vergleich von vorgeschlagenen programmierbaren Plattformen

In diesem Abschnitt stellen wir die in 3.7.1 und 3.7.2 präsentierten programmierbaren Knoten gegenüber. Für ein grobe Unterscheidung führen wir zwei Kategorien ein:

- Den möglichen *Ort*, an dem programmierbare Dienste in den beschriebenen Architekturen ausgeführt werden können.
- Die *Eigenschaften* der programmierbaren Plattformen.

Durch die unterschiedlichen Architekturen der vorgeschlagenen programmierbaren Plattformen kann man zunächst nach dem *Ort* der Ausführung von Diensten unterscheiden. Dies verfeinern wir im Detail weiter nach dem möglichen topologischen Ort im Netz, sowie dem möglichen Eingriff für Dienste in die jeweiligen Netzschichten. In Zusammenhang damit steht auch die konkrete Implementierung, ob Dienste im Kern bzw. im Userspace eines programmierbaren Knoten ausgeführt werden können.
Die *Eigenschaften* programmierbarer Knoten unterteilen wir in *Basiseigenschaften* (Flexibilität und Skalierbarkeit), *erweiterte Eigenschaften* (Sicherheit und Fehlertoleranz) und auch mögliche Erweiterungen bzw. *Adaptierbarkeit*.

In Abbildung 3.4 werden die beschriebenen Architekturen gegenübergestellt. Die vorgeschlagenen Architekturen zielen in erster Linie auf einen Einsatz am Netzrand, in *Edge-Routern* oder den Endsystemen selbst, ab. Alle beschriebenen Architekturen können ab der Internetschicht in den Datenstrom eingreifen. In einigen wenigen Architekturen ist es möglich in die

	Möglichkeit der Platzierung			Eingriff in Layer			Dienst Ausführung		Knoten: Basiseigenschaften				Erweiterte Eigenschaften			Adaptierbarkeit				
Kategorien / Programmierbare Knoten–Architekturen	Core–Router	Edge–Router, bzw. Gateway	Host	Layer 2	Layer 3/4	Application Layer	Kernbasiert	Userspace–basiert	OS–Flexibilität: Einbindung von weiteren Betriebssystemen	SW–Flexibilität: Einbindung anderer Sprachen/ Libraries	Skalierbarkeit: lokale Erweiterbarkeit	Skalierbarkeit: Rechenintensive Dienste	Security gegenüber internen Attacken	Security gegenüber externen Attacken	Fehlertolerante Basis (HW/SW Redundanz)	Einfache Integrierbarkeit in bestehende Router–Architekturen	Rapid Service Creation: Unabhängigkeit v. HW/ OS/ Sprache	Zusätzliche Integration von dedizierter Hardware	geringer Aufwand/ Kosten bei Änderungen	Integration von Grid Computing–Ressourcen
ANTS		X			X	X		X											X	
Switchware		X			X	X		X		X			X						X	
ANN (WUGS)	(X)	X			X	X		X	(X)	(X)	X	X			(X)		(X)			
PromethOS		X		(X)	X	X	X					(X)							X	
Snow on Silk		X			X	X		X		X			X						X	
Python		X			X	X		X											X	
AMnet	(X)	X	X	(X)	X	X	X	X		X	(X)	(X)	(X)	(X)		(X)	(X)	X	X	
FHiPPs		X			X	X		(X)			(X)	(X)	(X)					(X)		
Vorschlag dieser Arbeit	(X)	X			X	X		X	X	X	X	X	X	X	X	X	X	x	X	X

Abbildung 3.4: Programmierbare Knoten im Überblick

Network Interface-Schicht einzugreifen. Dies ist z.B. für den Einsatz auf programmierbaren WaveLan-Basisstationen (z.B. ReSoA [SRBW01]) von Vorteil.
Die Ausführung der Dienste, ob im Kern des Betriebssystems oder im Userspace, hat eine unmittelbare Auswirkung auf eine mögliche Kontrolle des Dienstes und damit auf die Sicherheit und Zuverlässigkeit der programmierbaren Plattform. Die präsentierten Ansätze sehen

zwar die Integration der in der Literatur beschriebenen, architekturunabhängigen Ansätze zur sicheren Ausführung von Diensten vor. Beim Entwurf der Architekturen der programmierbaren Knoten wurde der direkte Einfluss von Sicherheitsaspekten aber vernachlässigt.
Die *Basiseigenschaften* programmierbarer Knoten unterteilen wir in Flexibilität und Skalierbarkeit. Spezifische Anforderungen von Diensten machen es nötig, dass sie z.B. nur unter bestimmten Betriebssystemen, unter Einbindung dedizierter Software-Bibliotheken, angeboten werden können. Einige beschriebene Architekturen bieten zwar diese notwendige Flexibilität, im Gegenzug werden aber dann notwendige Sicherheitsmaßnahmen vernachlässigt. So widerspricht diesem Punkt z.B. die Verwendung einer *sicheren* Sprache für programmierbare Dienste. In dieser Arbeit schlagen wir eine Architektur vor, die Sicherheitsmechanismen und die notwendige Flexibilität zur Verfügung stellt.
Skalierbarkeit betrachten wir unter zwei Aspekten: Zum einen die Möglichkeit lokaler Erweiterungen, um z.B. parallel eine Vielzahl an Diensten anbieten zu können. Dies wird in den Architekturen nur am Rande betrachtet. Die präsentierten skalierbaren Vorschläge basieren auf der Verwendung dedizierter Hardware, was im allgemeinen mit hohen Kosten verbunden ist.
Zum anderen betrachten wir Skalierbarkeit unter dem Aspekt, ob rechenintensive Dienste (wie z.B. Transkodierung) zur Verfügung gestellt werden können. In diesem Zusammenhang spielen drei Punkte eine wesentliche Rolle: Die verwendete Programmiersprache, der Ausführungsort der Dienste und die Möglichkeiten lokaler Erweiterungen. Hierbei fällt ebenfalls auf, dass die beschriebenen Systeme rechenintensive Dienste nur mit dedizierter Hardware in skalierbarer Art und Weise zur Verfügung stellen können, was wiederum hohe Kosten zur Folge hat.
Die in dieser Arbeit vorgeschlagene Architektur stellt die notwendige Skalierbarkeit durch die Nutzung von Standardhardware zur Verfügung, was im allgemeinen einen Kostenvorteil nach sich zieht.

3.8 Zusammenfassung

In diesem Kapitel geben wir einen Überblick über den Stand der Technik. Zunächst erfolgt eine Übersicht über die Vielzahl an vorgeschlagenen und teilweise schon in Betrieb befindlichen Fehlertoleranzmechanismen. Grundlage für diese Arbeit sind folgende Punkte:

- Domänenübergreifende Fehlertoleranzmechanismen in der Internetschicht haben bei Ausfällen eine lange Konvergenzzeit, die für verschiedene Multimedia-Anwendungen nicht zu tolerieren ist. Die vorgeschlagenen Ansätze sind aber transparent gegenüber Applikationen.
- Transportschichtbasierte, fehlertolerante Overlay-Netze sind nicht transparent gegenüber Applikationen und in der Skalierbarkeit stark begrenzt. Außerdem berücksichtigen sie die Funktionalität der Applikationen nicht.
- Contentbasierte Overlay-Netze wurden ursprünglich entwickelt, um Daten im Internet skalierbar zu verteilen. Das Hauptaugenmerk liegt auf der Reduzierung der Last im Netz und auf dem ursprünglichen Server. Das zugrundeliegende Prinzip von redundantem, im Netz verteilten Content, wird in der vorliegenden Arbeit berücksichtigt.
- Es wurde ein Überblick gegeben über die vorgeschlagenen Mechanismen des Datentransports in Overlay-Netzen. Dieses Mechanismen erlauben keine (gegenüber Applikationen) vollkommen transparente Nutzung der Overlay-Netze. In der vorliegenden Arbeit wird ein transparenter Datentransport in Overlay-Netzen vorgeschlagen.
- Es wurde ein Überblick über Fehlertoleranzmechanismen in Endsystemen gegeben. Für die vorliegende Arbeit von Relevanz ist eine mögliche Interaktion, bzw. die Einbindung der Mechanismen.

Außerdem wird ein Überblick über programmierbare Knotenarchitekturen gegeben. Die in der Literatur beschriebenen Architekturen behandeln das Thema Fehlertoleranz bzw. Sicherheit und Skalierbarkeit der Architektur nur am Rande. Der in dieser Arbeit vorgestellte Ansatz verbessert entsprechende Punkte.
In den folgenden Kapiteln beschreiben wir einen neuen, flexiblen, netzbasierten, schichtenübergreifenden Fehlertoleranzmechanismus, der Applikationen direkt unterstützt. Wir schlagen vor, diesen Fehlertoleranzmechanismus als ladbaren Dienst, mit programmierbaren Plattformen zur Verfügung zu stellen.

Kapitel 4

Mehrdimensionale adaptive Fehlertoleranz

In Kapitel 3 wurden die wesentlichen Fehlertoleranzmechanismen im Internet erläutert. Ein Grundproblem diese Arbeit betreffend, ist die in Abschnitt 3.2 präsentierte, lange Konvergenzzeit von netzübergreifenden Restaurationsverfahren bzw. Rerouting in der Internetschicht. Jüngste Vorschläge zur Lösung dieses Problems führen Overlay-Netze über der Internetschicht ein, die durch redundante Wege entsprechende Fehler im Netz behandeln können. Abhängig vom angewandten Verfahren innerhalb der Overlay-Netze, können Fehler im Netz vollkommen maskiert werden durch schnelle Ersatzschaltverfahren (z.B. 1+1 Ersatzschalten mit redundanter Übertragung), bzw. die Konvergenzzeit durch Restauration im Overlay-Netz deutlich herabgesetzt werden.
Die beschriebenen Ansätze (siehe Abschnitt 3.3.1) haben dennoch drei wesentliche Defizite:

- Begrenzte Skalierbarkeit
- Kein transparenter Betrieb gegenüber Applikationen möglich
- Fokus auf den Punkt-zu-Punkt Datentransport, keine direkte Unterstützung der Funktionalität von verteilten Applikationen

Die beschriebenen fehlertoleranten, transportschichtbasierten Overlay-Netze sind in der Skalierbarkeit begrenzt. Im Wesentlichen zielen sie auf einen, an Teilnehmern oder der Verbreitung, begrenzten Einsatz (z.B. einem Intranet) ab.
Darüber hinaus können die beschriebenen Ansätze nicht ohne entsprechende Anpassungen der Endsysteme eingesetzt werden. Ein transparenter Betrieb der fehlertoleranten Overlay-Netze ist somit nicht möglich. Dies stellt vor allem Netzbetreiber vor Probleme, da sie nicht ohne weiteres die Fehlertoleranz des Netzes erhöhen können, ohne entsprechende Konfiguration der Teilnehmer.
Des Weiteren konzentrieren sich die in Abschnitt 3.3.1 beschriebenen Ansätze ausschließlich auf den fehlertoleranten Ende-zu-Ende-Datentransport in Unicast-Verbindungen. Eine Unterstützung von Anwendungen der Gruppenkommunikation ist nicht vorgesehen. Dies widerspräche auch dem Problem der unzureichenden Skalierbarkeit der Ansätze. Hauptaugenmerk der in den Abschnitt 3.3.2 vorgestellten Ansätze ist die ressourcenoptimierte Verteilung von Daten im Netz. Sowohl Content Distribution Networks als auch kooperierende Caches bieten

prinzipiell die Möglichkeit, Content mehrfach, und damit redundant, im Netz zu platzieren. Damit sind diese Ansätze eine gute Erweiterung für einen Fehlertoleranzmechanismus. Das eigentliche Ziel bestimmter Applikationen, z.B. die Darstellung eines bestimmten Contents, kann durch diese Ansätze unterstützt werden. Die beschriebenen Ansätze unterliegen aber prinzipiell zwei Einschränkungen:

- Abhängig vom Content kann eine Zwischenspeicherung im Netz u.U. unmöglich sein.
- Kosten- Nutzen-Verhältnis.

Handelt es sich z.B. um verschlüsselte Kommunikationsverbindungen, so ist eine Zwischenspeicherung im Netz nicht möglich bzw. aus Sicherheitsgründen auch nicht gewünscht. Auch für hochdynamische Daten, wie z.B. den gegenwärtigen Kurswert einer Aktie an der Börse, ist eine Zwischenspeicherung im Netz nur wenig sinnvoll.
Auch das Kosten- Nutzen-Verhältnis spielt eine gewisse Rolle. So erscheint es u.U. ökonomisch wenig sinnvoll einen bestimmten Content zu replizieren, der nur sehr selten angefragt wird. Im Allgemeinen wird durch CDN daher nur populärer Content angeboten.
Abgesehen von diesen Einschränkungen, können content-basierte Overlay-Netze aber sehr wohl zur Unterstützung der Fehlertoleranz von Applikationen benutzt werden.

Aus diesem Grund schlagen wir in dieser Arbeit ein transportschichtbasiertes, fehlertolerantes Overlay-Netz vor, das schichtenübergreifend auch den Content berücksichtigt. Somit geht der vorgeschlagene Mechanismus direkt auf die spezifischen Eigenschaften der jeweiligen Applikationen ein.
Dieses Kapitel führt Fehlertoleranz als Dienst im Netz mit den folgenden Eigenschaften ein:

- **Domänenübergreifend bzw. netzübergreifend**: Der Fehlertoleranzdienst kann über (heterogene) administrative Domänen hinweg Fehler maskieren.
- **Schichtenübergreifend**: Der Fehlertoleranzdienst kann über verschiedene Schichten hinweg Fehler maskieren bzw. abschwächen.
- **Adaptiv**: Der Fehlertoleranzdienst ist relativ einfach an neue Applikationen anpassbar.

Zunächst stellen wir das für diese Arbeit zugrundeliegende System- bzw. Fehlermodell vor. Wesentlich für den Beitrag der Arbeit ist der eingeführte *Applikationsfehler*.
Anschließend präsentieren wir das Modell der adaptiven Fehlertoleranz. Wesentlich ist die Behandlung von Applikationsfehlern durch horizontale, vertikale und longitudinale Fehlermaskierung bzw. Fehlerabschwächung. Diese Basismodule werden an die jeweiligen Anforderungen adaptiert. Des Weiteren betrachten wir die Einbettung des Fehlertoleranzdienstes in die Ende-zu-Ende-Kommunikation.
Nach einem Überblick über die Komponenten präsentieren wir die adaptive Fehlertoleranz als Modell eines programmierbaren Netzdienstes. Wesentlich ist hier die Unterteilung in einen ladbaren Dienst und die zugrundeliegende, programmierbare Plattform.

4.1 System- und abstrahierte Fehlermodellierung

In diesem Abschnitt gehen wir auf die Anforderungen von Internetapplikationen an die Kommunikationsinfrastruktur bzw. deren Fehlertoleranzmechanismen ein. Zunächst beschreiben wir in allgemeinen Systemmodellen die Anforderungen an Endgeräte und an ein Netz. Anschließend verwenden wir diese allgemeinen Modelle als Bausteine für applikationsspezifische System- bzw. Fehlermodelle. Am Beispiel einer Client-Server-Kommunikation und einer einfachen Gruppenkommunikation werden applikationsspezifische Anforderungen präsentiert. Des Weiteren führen wir den Begriff des Applikationsfehlers ein, dessen Behandlung das Ziel dieser Arbeit ist.

4.1.1 Fehler in den Endgeräten

In Abbildung 4.1 sind unterschiedliche Möglichkeiten für Fehler angeführt, die lokal im Rechner bzw. Endgerät, ihre Ursache haben. In dieser Arbeit unterteilen wir Fehlerzustände in Endsystemen in drei Kategorien:

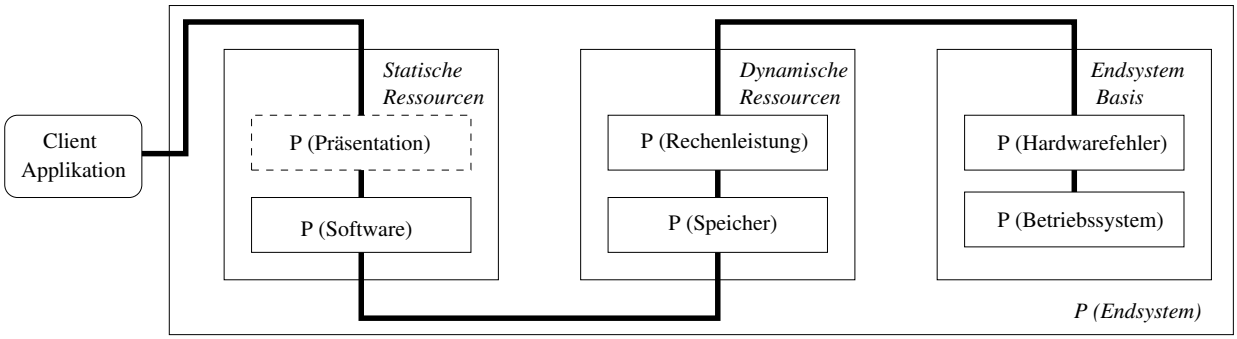

Abbildung 4.1: Möglichkeiten für lokale Fehler

- Fehler in der *Basis* des Endsystems
- Fehler in *dynamischen* Ressourcen
- Fehler in *statischen* Ressourcen

Fehler in der Basis des Endsystems bezeichnen zum einen Hardwarefehler, als auch zum anderen Softwarefehler im Betriebssystem des Endgerätes.
Fehler durch Mangel an *dynamischen* Ressourcen sind durch die momentane Auslastung im System gekennzeichnet. So kann es sein, dass zum Zeitpunkt, zu dem eine Applikation ausgeführt werden soll, nicht genügend Arbeitsspeicher zur Verfügung steht, oder nicht genug Rechenleistung vorhanden ist, weil das Endgerät z.B. parallel andere Prozesse bearbeitet.
Eine weitere Ursache für Fehler kann darin liegen, dass gewisse *statische Ressourcen* auf dem Endsystem nicht zur Verfügung stehen. Damit fassen wir Fehler zusammen, die letztlich in direkter Beziehung zu den gewünschten Applikationen stehen. So kann es z.B. sein, dass auf einem Endsystem eine bestimmte Applikation nicht lauffähig ist, da bestimmte Bibliotheken nicht vorhanden sind. Ein weiteres Beispiel wäre die unzureichende Voraussetzung für die

Präsentation einer Applikation. So kann z.B. die Auflösung des Displays im Endgerät nicht genügen, oder eine benötigte Soundkarte nicht vorhanden sein.

Im Rahmen dieser Arbeit betrachten wir, die Endsysteme betreffend, in erster Linie Fehlerzustände aufgrund von Mängeln in statischen Ressourcen. In diesem Zusammenhang gehen wir ebenfalls kurz auf Fehlerzustände ein, die durch Mängel in dynamischen Ressourcen entstehen. Der Grundgedanke dieser Arbeit ist die netzseitige Unterstützung von Endgeräten bei der Ausführung von Applikationen. Aus diesem Grunde führen wir ein flow- bzw. applikationsbasiertes Systemmodell von Endsystemen ein.
Fehlerursachen, die Basis der Endsysteme betreffend, werden im Weiteren nicht behandelt. Eine netzseitige Unterstützung würde bei dieser Art von Fehlern keinen großen Vorteil bringen. Hardwarefehler sollten direkt im Endsystem behandelt werden. Fehler im Betriebssystem der Endgeräte können z.B. durch ein entsprechendes Update behandelt werden.

4.1.2 Fehler im Netz

In Abschnitt 4.1.1 werden Fehlerursachen in Endsystemen kurz betrachtet, die letztlich zu Fehlerzuständen in Applikationen führen. In diesem Abschnitt gehen wir auf Fehler bei der Datenübertragung im Netz ein. Ziel ist die Darstellung der Zusammenhänge von Fehlerzuständen bzw. Ausfällen im Netz, die zu Fehlerzuständen in Applikationen führen können. Im Unterschied zu den Endsystemen ist eine detaillierte Betrachtung von Fehlerursachen im

Observiertes Verhalten (von Aussen)		Mögliche Ursache im Netz
Manipulation von Paketen:	Daten verändert, zusätzliche Pakete	(Bitfehler), aktiver Angriff
Paketverlust:	Verzögerung ausserhalb Toleranz	Überlast, Routingprobleme (Loops)
	statistisch	Überlast im Netz: Überlauf von Warteschlangen
	kontinuierlich	(Teilweiser) Ausfall von Infrastruktur (HW/SW)

Abbildung 4.2: Fehlerquellen bei der Datenübertragung

Netz, aufgrund der möglichen Komplexität von IP-WAN, hier nicht zielführend.
Wir betrachten daher nicht die Ursachen für Fehler im Netz, sondern lediglich die Fehlerzustände bzw. Ausfälle im Netz. In Abbildung 4.2 sind die für diese Arbeit relevanten Fehlerzustände bzw. Ausfälle des Netzes während der Datenübertragung angeführt.
So können Datenpakete auf dem Weg zwischen den Kommunikationspartnern durch das Netz unabsichtlich verändert bzw. manipuliert werden. Im Laufe der Zeit wurden diese klassischen Bitfehler aber immer seltener. Durch die fortschreitende Entwicklung der Systeme, in Kombination mit geeigneten Korrekturmechanismen, kommt dies in heutigen Systemen, mit Ausnahme des Mobilfunks, praktisch nicht mehr vor. In diesem Zusammenhang trat in den letzten Jahren hingegen das Thema Sicherheit immer mehr in den Vordergrund. So sind Angriffe dritter Parteien im Netz denkbar, die Datenpakete absichtlich manipulieren bzw. zusätzliche, an die Applikation angepasste Datenpakete, zur Störung in das Netz senden. Derartige Angriffe können die betroffenen Applikationen effektiv stören. In Denial of Service-Attacken kann z.B. die Kommunikation vollständig zum Erliegen kommen. Die

relativ junge Forschung auf diesem Gebiet schlägt einige Ansätze vor (z.B. in [HJS03]), wie diesem Problem von Seiten des Netzes aus begegnet werden kann.
In dieser Arbeit betrachten wir Fehlerzustände in Applikationen, deren Ursache auf Angriffe dritter Parteien zurückgeht, nicht weiter. Für unsere Dienstarchitektur von Relevanz ist lediglich eine mögliche Interaktion mit den vorgeschlagenen Sicherheitsmechanismen.
Wir fokussieren uns auf die Auswirkungen von Paketverlusten auf Applikationen und die applikations- bzw. situationsabhängige Behandlung daraus resultierender möglicher Fehler. Wir unterscheiden statistische und kontinuierliche Verluste von Paketen.

Mit *statistischen Verlusten* ist gemeint, dass Pakete eines Datenstroms scheinbar willkürlich beim Empfänger ankommen oder verloren gehen. Gemäß einer a posteriori abschätzbaren Wahrscheinlichkeit gehen einzelne Pakete im Netz verloren. Dies kann z.B. dann der Fall sein, wenn sich bestimmte Warteschlangen in einem Router füllen, und u.U. neu ankommende Pakete vom Router verworfen werden müssen. Je nach Lastsituation werden einige Pakete eines bestimmten Datenstroms somit statistisch verworfen. Im Netz muss es somit, im eigentlichen Sinn, zu keinem Fehler bzw. Ausfall gekommen sein. Alle Elemente des Netzes funktionieren fehlerfrei. Übersteigt aber eine situationsabhängige Last die Kapazität des Netzes, kommt es zu statistischen Paketverlusten. An diesem Punkt setzen Quality of Service- (QoS-) Netzarchitekturen an. Im Allgemeinen ist es das Ziel von QoS-Mechanismen statistische Verluste oder Verzögerungen von Datenpaketen, angepasst an die Anforderung von Applikationen, zu verhindern bzw. zu minimieren. Dies kann zum einen durch eine differenzierte Behandlung von Datenströmen im Netz bewerkstelligt werden, zum anderen durch Zugangskontrollverfahren bzw. durch Reservierungen von Ressourcen im Netz. Wichtig im QoS-Kontext ist eine durchgängige Behandlung der Datenströme. Wie in Abbildung 2.1 skizziert, besteht das Internet aus einer Vielzahl an administrativ eigenständigen Domänen. Tritt ein statistischer Paketverlust in einer Domäne auf, die über keinen QoS-Mechanismus verfügt, dann haben eventuell in anderen Domänen vorhandene QoS-Mechanismen keine Möglichkeit einzugreifen.

Mit *kontinuierlichen Paketverlusten* sind Verluste von Paketen gemeint, die auf einen stationären Fehlerzustand bzw. Ausfall im Netz zurückzuführen sind. Die Ursachen dafür können vielfältig sein. So können u.U. einzelne administrative Domänen vollständig ausfallen (z.B. durch einen Terrorakt). Ein weiteres Beispiel wäre ein Fehler in den Routingprotokollen zwischen den administrativen Domänen.

Die vorliegende Arbeit fokussiert sich, Fehler im Netz betreffend, auf die Behandlung von Fehlerzuständen in Applikationen, die auf *kontinuierliche Paketverluste* im Netz zurückzuführen sind. Der vorgeschlagene, netzbasierte Fehlertoleranzdienst soll diese Fehlerzustände des Netzes gegenüber Applikationen maskieren.
Der Vorschlag dieser Arbeit steht nicht in Konkurrenz zu QoS-Architekturen. Durchgängige QoS-Architekturen fokussieren sich auf die Behandlung von statistischen Paketverlusten. Die vorgeschlagenen Architekturen setzen auf ein funktionierendes Routing auf [Gla03]. Darauf bauen dann vorgeschlagene QoS-Mechanismen auf und reservieren z.B. entsprechende Netzressourcen.
Kommt es nun beispielsweise zu einem, wie in Abschnitt 3.2.2 beschriebenen, Rerouting durch BGP, bieten QoS-Mechanismen keinen Schutz. Nach der Konvergenz des Rerouting müssen auch innerhalb der QoS-Architekturen die Ressourcen neu reserviert werden. Erst

nach dem Abschluss der Restauration steht der QoS-Mechanismus dem Netz wieder zur Verfügung.
Ähnlich wie im Fall der Gruppenkommunikation sind QoS-Mechanismen aber bis dato im Internet nicht umgesetzt. Die wesentlichen technischen Probleme scheinen auch in diesem Fall seit längerem gelöst. Eine Umsetzung der Lösungen hat praktisch aber kaum stattgefunden. Nur in einzelnen abgeschlossenen Systemen, wie z.B. Intranetzen, werden QoS-Architekturen verwendet. Mögliche Gründe für die quasi nicht vorhandene Umsetzung sehen wir ähnlich wie für den Fall der Gruppenkommunikation. Eine große Zahl an Providern müsste eng kooperieren, um durchgängige Lösungen anbieten zu können. Dies erweist sich in der Praxis anscheinend als große Hürde. Ob sie genommen werden kann, wird sich in den nächsten Jahren zeigen.
Einstweilen wäre es möglich, den in dieser Arbeit beschriebenen, Dienst als eine Art Zwischenlösung zu verwenden bis durchgängige QoS angeboten werden kann. Ansätze wie z.B. OverQos [SSBK04] müssten dann integriert werden. Dies ist aber nicht das Ziel dieser Arbeit, sondern eine mögliche Erweiterung.

Im Folgenden werden im Wesentlichen nur kontinuierliche Paketverluste berücksichtigt. Verzögerungen bzw. Manipulationen von Paketen in den Datenströmen werden nicht weiter betrachtet. Wesentlich für diese Arbeit ist auch die Sichtweise auf ein Netz. Wir betrachten im Systemmodell das Netz nicht aus der Sichtweise aller übertragenen Datenströme, sondern aus der Sichtweise eines bestimmten Datenstroms. Somit betrachten wir das Netz flow-basiert. Hintergrund für die flow-basierte Sichtweise ist das, an die Bedürfnisse einer jeweiligen Applikationen, angepasste Verhalten des Netzes durch den vorgeschlagenen Fehlertoleranzdienst.

4.1.3 Fehlermodell für Client-Server-Kommunikation

In den Abschnitten 4.1.1 und 4.2.1 sind die für diese Arbeit zugrundeliegenden Systemmodelle von Netzen und Endsystemen allgemein dargelegt. In diesem und dem nächsten Abschnitt fügen wir, aufbauend auf diese Bausteine, exemplarisch Systemmodelle von verteilten, netzbasierten Applikation zusammen. Im Folgenden beziehen wir uns auf eine herkömmliche Client-Server-Kommunikation über das Internet, und führen ein applikationsspezifisches System- bzw. Fehlermodell ein.
In einer einfachen Client-Server-Kommunikation sind zwei Endsysteme beteiligt, die über ein Netz Daten austauschen. Dies ist in Abbildung 4.3 vereinfacht dargelegt.

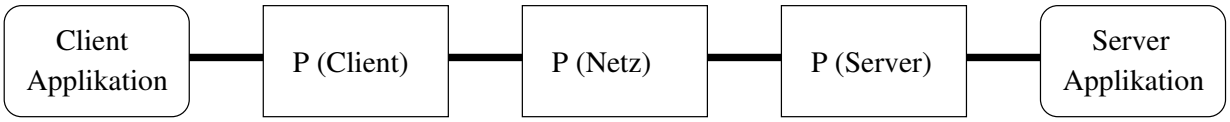

Abbildung 4.3: Fehlermodell Client-Server

Um einen entsprechenden Fehlertoleranzmechanismus an die Applikation anzupassen, ist eine detaillierte Kenntnis des Systems notwendig. Daher müssen zur genaueren Beschreibung im Einzelfall einige applikationsabhängige Verfeinerungen vorgenommen werden. Zum

einen spielt das verwendete Protokoll für den Datentransport eine Rolle. Die Kommunikation kann unidirektional bzw. bidirektional vonstatten gehen. Der Datentransport kann verlässlich oder unzuverlässig organisiert sein.
Betrachtet man auf Seiten des Servers die zur Verfügung gestellten Daten, stellt sich zum anderen die Frage, ob es sich um Daten bzw. Dokumente handelt, die dynamisch (u.U. in Realzeit) erzeugt worden sind (z.B. ein live streaming), oder ob es sich um statische Dokumente handelt, die man auch an einer anderen Stelle im Netz positionieren könnte, wie z.B. Filme aus einer "Video on Demand-Videothek".
Abgesehen von diesen allgemeinen Eigenschaften der jeweiligen Applikation, muss in das Systemmodell noch eine weitere Größe Einfluss finden: Die applikationsabhängigen Anforderungen an Endsysteme und das Netz. Auf Seiten des Endsystems sind dies im Wesentlichen die notwendigen lokalen Ressourcen, auf Netzseite die in Abschnitt 4.2.1 beschriebenen Eigenschaften.
So kann es z.B. sein, dass im Falle einer Video on Demand-Applikation die Anforderungen an den Server, aus Sicht der Fehlertoleranz, eher gering sind. Ein gewünschter Film könnte u.U. von einer Vielzahl von verteilten Servern bzw. Caches bezogen werden. Somit würden Paketverluste, in der Kommunikation mit einzelnen Servern, eher eine untergeordnete Rolle spielen. Für den Fall, dass es nicht möglich ist, den gewünschten Film von einem anderen Ort im Netz zu beziehen, sind die Anforderungen an das Netz und den Server entsprechend höher.
An diesem einfachen Beispiel sehen wir anschaulich die spezifischen Anforderungen von Applikationen an die Infrastruktur, bzw. an deren Fehlertoleranzmechanismen.

4.1.4 Fehlermodell für Gruppenkommunikation

In Abschnitt 4.1.3 wurde das applikationsabhängige System- bzw. Fehlermodell für eine einfache Client-Server-Applikation dargelegt. Das beschriebene, auf den Einzelfall anzupassende, Systemmodell des Gesamtnetzes beschränkt sich auf die Ende-zu-Ende-Sichtweise. Fehlerzustände im Netz werden durch das in Abschnitt 4.2.1 dargelegte Modell beschrieben. Die Fehlerursache wird vollständig abstrahiert.

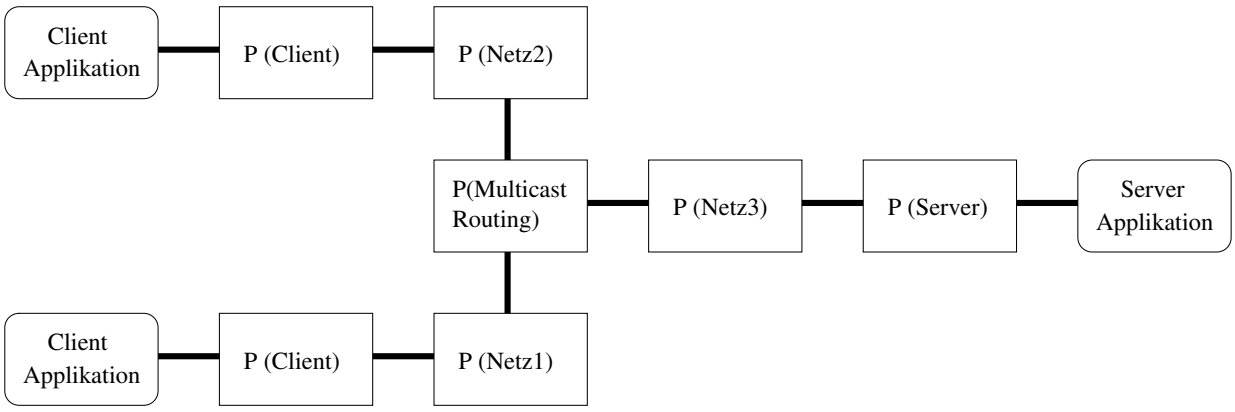

Abbildung 4.4: Exemplarisches Fehlermodell für eine Gruppenkommunikation

In diesem Abschnitt stellen wir das System- bzw. Fehlermodell für eine Anwendung der Gruppenkommunikation dar. In diesem Fall ist zusätzlich, zu der für den Fall der Client-Server-Kommunikation nötigen Infrastruktur im Netz, noch ein Netzdienst notwendig, der den Multicast-Datentransport zur Verfügung stellt. Wir bauen das Systemmodell der Gruppenkommunikation unabhängig von der eigentlichen Implementierung dieses Netzdienstes auf, wie in Abbildung 4.4 exemplarisch skizziert.
Im Beispiel beschreiben wir das einfachste Szenario für eine Gruppenkommunikation. Ein Server sendet einen Datenstrom in das Netz. Zwei Empfänger empfangen diesen Strom. Im Beispiel nehmen wir an, dass der Datenstrom vom Server über ein Teilnetz zu einem Multicast-Router gesendet wird. Dort wird der Datenstrom dupliziert, und ebenfalls wieder über Teilnetze an die Empfänger weitergeleitet.
Auch für den Fall der Gruppenkommunikation ist im Einzelfall wieder eine genauere, applikationsabhängige Modellierung des Systems notwendig. So können z.B. mehrere Empfänger und auch mehrere Multicast-Router beteiligt sein. Die Multicast-Router können, für den Fall des Application Layer Multicasting (siehe Abschnitt 2.2.3), in den Empfängern alloziert sein, oder wie im Beispiel, eigenständige Netzelemente sein. Des Weiteren kann es zu Hierarchien in den Multicast-Bäumen kommen.

Zusätzlich zu diesen Betrachtungen müssen für ein detaillierteres Systemmodell auch noch spezielle Anforderungen von Applikationen berücksichtigt werden. So gibt es z.B. unterschiedliche Realzeitanforderungen von Applikationen an die Gruppenkommunikation:
In einer Konferenz mit mehreren Teilnehmern, die auf Multicast-Technologien aufsetzen, spielt die Verzögerung eine große Rolle. Im Falle eines Multicast-Szenarios, bei dem ein Film an viele Empfänger gesendet wird, sind die Anforderungen gegenüber Fehlertoleranzmechanismen u.U. geringer. Die Daten können z.B. lokal gepuffert werden. Über fehlertolerante Multicast-Protokolle könnten verloren gegangene Daten u.U. erneut angefordert werden können, bevor sie abgespielt werden. Ein weiteres Beispiel sind push-gesteuerte Software-Updates auf Endsystemen über Multicast. Paketverzögerungen bzw. Verluste spielen auch in diesem Fall eine untergeordnete Rolle.
Diese angeführten Beispiele für Anwendungen der Gruppenkommunikation können alle die gleiche Netzinfrastruktur benutzen.
Des Weiteren muss berücksichtigt werden ob, ähnlich wie im Fall der Client-Server-Kommunikation, die von Servern ausgesendeten Daten a priori dupliziert, und redundant im Netz auf mehreren Server bereitgestellt werden können.

Wir sehen, dass die applikationsspezifischen Anforderungen an die Fehlertoleranz, ähnlich wie im Fall der Client-Server-Kommunikation, stark variieren können.

Für den Fall der Gruppenkommunikation kommt gegenüber dem Fall der Client-Server-Kommunikation aber noch eine weitere *Dimension* ins Spiel. Im Falle der Client-Server-Kommunikation sind die Ziele von Client und Server identisch: Die angeforderten Daten sollen gemäß Anforderungen auf dem Client entsprechend verarbeitet werden können. Dieses Ziel haben sowohl Client als auch Server.
Für den Fall der Gruppenkommunikation hat der Server das Ziel, dass Daten auf mehreren Empfängern empfangen und verarbeitet werden können. Der einzelne Empfänger hat zunächst aber nur das primäre Ziel, dass er selbst die Daten empfängt und entsprechend verarbeiten kann.

Diese Betrachtung gilt in jedem Fall für den netzbasierten Multicast (z.B. IP-Multicast). Für den Fall des Application Layer Multicast gilt dies erst nach einer logischen Trennung zwischen Multicast-Routing und der eigentlichen Verarbeitung der Daten, was in der Praxis parallel auf den Endsystemen läuft. Auch in diesem Punkt muss wieder applikationsabhängig unterschieden werden. Für bestimmte Anwendungen der Gruppenkommunikation kann ein Empfänger durchaus noch das sekundäre Ziel haben, dass auch andere Empfänger die Daten erhalten. Dies kann z.B. bei einer Konferenzschaltung der Fall sein.

Diese u.U. unterschiedlichen Ziele von Sender und Empfängern können somit Auswirkungen auf die Anforderungen an Fehlertoleranzmechanismen haben. Fordert ein Sender Fehlertoleranzmechanismen im Rahmen einer Gruppenkommunikation mit Empfängern, dann kann dies ein weit umfassenderer Netzdienst sein, als wenn z.B. nur ein Empfänger Fehlertoleranzdienste vom Netz fordert.
Im Beispiel bedeutet dies konkret: Für den Fall, dass der Sender Fehlertoleranzmechanismen anfordert, müsste das in Abbildung 4.4 dargestellte Gesamtsystem fehlertolerant sein. Fordert jedoch nur ein Empfänger Fehlertoleranzmechanismen, dann muss lediglich ein Teil des Gesamtsystems fehlertolerant sein.

4.1.5 Einführung des *Applikationsfehlers*

Die in den Abschnitten 4.1.1, 4.1.3 und 4.1.4 dargelegten spezifischen Fehlermodelle fassen wir in diesem Abschnitt zusammen. Ziel dieser Arbeit ist ein Fehlertoleranzmechanismus, der ein Versagen von Applikationen verhindert, durch Maskierung bzw. Abschwächung von Fehlerzuständen in Endsystemen oder bei der Datenübertragung.
Im Rahmen dieser Arbeit führen wir somit folgende Definition des *Applikationsfehlers* ein:

- Als Fehler definieren wir eine negative Veränderung des Verhaltens einer netzbasierten Applikation vom Gutzustand, die von der Applikationsschicht eines Kommunikationsteilnehmers wahrgenommen wird, und gegenüber einem Benutzer, seitens der Applikation, nicht maskiert werden kann.
- Das Augenmerk liegt dabei auf der Maskierung kontinuierlicher Paket- bzw. Datenverluste, die nicht auf Angriffe im Netz zurückzuführen sind, sondern auf Fehlerzustände im Netz.

Wir gehen von dem Fall aus, dass ein allgemeiner Benutzer (ein Mensch oder eine Maschine) eine netzbasierte Applikation über eine Schnittstelle startet. Ein Applikationsfehler tritt dann auf, wenn ein auftretender Fehler von diesem Interface nicht maskiert werden kann, vom Benutzer aber bemerkt würde.

Ein Beispiel hierfür wäre die Übertragung von Videodaten zwischen einem Sender und einem Empfänger. Schon während der Übertragung beginnt der Empfänger zeitverzögert mit dem Abspielen, und füllt zunächst seinen Eingangspuffer. Während des Abspielens kommt es nun zu einem Ausfall bei der Datenübertragung (mögliche Gründe: Probleme im Netz, Serverausfall). Dauert dieser Ausfall länger, als die im Eingangspuffer zur Verfügung stehenden Datenpakete für das Abspielen benötigen, kann ein Versagen gegenüber dem Benutzer nicht maskiert werden, es liegt ein Applikationsfehler vor.

Diese Arbeit schlägt somit einen Fehlertoleranzmechanismus vor, der Applikationsfehler gegenüber Benutzern maskiert. Im Folgenden beschreiben wir das zugrundeliegende Modell der mehrdimensionalen, adaptiven Fehlertoleranz.

4.2 Modell der universellen adaptiven Fehlertoleranz

In diesem Abschnitt wird ein neuer adaptiver Fehlertoleranzmechanismus vorgeschlagen, der an das heterogene Umfeld des Internets angepasst ist. Schichten- und domänenübergreifend werden *Applikationsfehler* (siehe Abschnitt 4.1.5), auf jeweilige Applikationen zugeschnitten, maskiert.
Zunächst führen wir drei Kategorien von Fehlern ein. Für die Maskierung dieser Kategorien schlagen wir Basismodule vor, welche die entsprechenden Kategorien von Fehlern maskieren. Außerdem geben wir einen Überblick über die Dienstarchitektur.

4.2.1 Anforderungen von Applikationen an die Fehlertoleranz

Ziel dieser Arbeit ist ein Fehlertoleranzmechanismus, der Applikationen direkt unterstützt. Die in den Abschnitten (4.2.1, 4.1.3 und 4.1.4) beschriebenen Ursachen für Applikationsfehler betrachten wir aus der Sichtweise der Applikationen bzw. möglicher Fehlertoleranzmechanismen. In Abbildung 4.5 sind die Anforderungen von Applikationen an Fehlertoleranzme-

Fehler Kategorien / Eigenschaften	Statischer Content (replizierbar)	Content nicht replizierbar	Ausfall bzw. Fehlerzustand: Content/ Applikation	Ausfall bzw. Fehlerzustand: Datentransport	Ausfall bzw. Fehlerzustand: Endgerät statische/ dynamische Ressourcen	Mögliche Ursache	Fehlertoleranz Maßnahme	Wissen über Applikationsschicht
Kategorie 1	X		X			z.B. Server Reboot/ Überlast	Maskierung v. Contentausfall	notwendig
Kategorie 2		X		X		z.B. Rerouting	alternativer Pfad in Overlay−Netz	nicht notwendig
Kategorie 3	unabhängig				X	z.B. falscher Codec im Endsystem	Content−Anpassung z.B. Transkodierung	notwendig
Kategorie 1+3	X		X		X	Kombination aus 1+3	Redundanter adaptierter Content	notwendig
Kategorie Realzeit/ statistisch		X		statistisch		Überlast im Netz	Alternative Pfade & adaptierter Content	notwendig

Abbildung 4.5: Applikationsfehler, unterschieden nach abstrahierter Fehlerursache

chanismen in drei Kategorien klassifiziert. Diese konzentrieren sich auf Applikationsfehler kontinuierlicher Art. Mögliche Ursachen dafür sind Ausfälle auf Seiten des Servers, beim Datentransport oder ein Versagen in den Ressourcen der Endsysteme. Aus Benutzersicht wird, unabhängig von den Ursachen, ein Versagen der Applikation bemerkt. Wie exemplarisch angeführt, sind auch Kombinationen aus den vorgeschlagenen Kategorien möglich.
Die erste Kategorie von Applikationsfehlern hat als abstrahierte Fehlerursache den Ausfall eines angeforderten Content. Dafür kann es mehrere Gründe geben (wie z.B. den Ausfall der Serversoftware, Hardwarefehler, Wartungsarbeiten etc.). Wir subsumieren die diversen Möglichkeiten unter dem Begriff: *Contentausfall*.
Die zweite Kategorie beschränkt sich auf Ausfälle während des Datentransports durch das Internet zwischen den Kommunikationspartnern. Auch hier abstrahieren wir mögliche Fehlerursachen (z.B. Rerouting, Ausfälle von Domänen, Ausfall einer Multicast-Datenverteilung).

Wir fassen sie unter dem Begriff: *kontinuierlicher Ausfall beim Datentransport* zusammen. Die dritte Kategorie von Applikationsfehlern schließlich umfasst Inkompatibilitäten zwischen den Gegebenheiten bzw. Voraussetzungen in den Endsystemen und den gewünschten Eigenschaften von Applikationen, und dem daraus resultierenden Versagen.

Die **Basisanforderungen** an den in dieser Arbeit präsentierten Fehlertoleranzmechanismus sind die Behandlung der beschriebenen Kategorien von Applikationsfehlern: namentlich Maskierung von Contentausfällen, Adaptierung von Content und die zur Verfügungstellung von alternativen Pfaden.

Zusätzlich zu diesen geforderten grundlegenden Eigenschaften ist ferner wichtig, dass der Fehlertoleranzmechanismus einfach und schnell adaptierbar ist hinsichtlich:

- Topologischer Eigenschaften der zu schützenden Applikation
- Spezifischer Anforderungen von Applikation (z.B. Codec)

Diese **erweiterten Anforderungen** an einen Fehlertoleranzmechanismus liegen im möglichen schnellen Wandel der Applikations-Landschaft im Internet begründet.

Die angeführten Kategorien 1 bis 3 unterscheiden Applikationen nicht nach deren Realzeitanforderungen. Wir beschränken die Betrachtung auf diejenigen Applikationen, die nicht durch in Abschnitt 3.2 beschriebene Fehlertoleranzmechanismen ausreichend geschützt werden können.
Wir gehen im Rahmen dieser Arbeit von einem kontinuierlichen Applikationsfehler aus, der maskiert werden soll. *Somit beschränken wir uns ausschließlich auf zu schützende Applikationen, die gewisse Anforderungen an Verzögerungen haben.* Ein Beispiel für zu schützende Applikationen ist die Anfrage nach einer bestimmten Website durch einen Benutzer. Ist diese Anfrage nur durch die im Internet vorhandenen Mechanismen geschützt, kann es bei einem Ausfall zu Verzögerungen im Bereich von Minuten kommen. Dies wäre für den Benutzer bzw. die entsprechende Applikation nicht tolerierbar.
Hingegen würde beim Transfer von E-Mails durch das Internet, eine durch einen Ausfall bedingte entsprechende Verzögerung, in der Regel nicht bemerkt werden.

An dieser Stelle gehen wir noch kurz auf eine Kategorie von Applikationsfehlern ein, die auf statistische Paketverluste zurückzuführen ist. Wir gehen von Applikationen aus, die hohe Realzeitanforderungen an die Datenübertragung stellen, ein Beispiel wäre VoIP. Für derartige Applikationen deckt *Kategorie 2* Applikationsfehler ab, die durch kontinuierliche Paketverluste begründet sind.
Da aber in diesem Zusammenhang auch statistische Paketverluste eine Rolle spielen (verloren gegangene Pakete können nicht erneut verschickt werden), gehen wir kurz auf diese ein. Die in Abschnitt 4.2.1 kurz angesprochenen QoS-Architekturen minimieren statistische Paketverluste. Da diese Architekturen aber derzeit nicht netzübergreifend im Internet umgesetzt sind, könnte der Beitrag dieser Arbeit als Zwischenlösung genutzt werden, wie in Abschnitt bereits angeführt.

4.2.2 Modulare Basisfunktionalität

In Abbildung 4.6 stellen wir das aus den Anforderungen von Applikationen (siehe Abschnitt 4.2.1) abgeleitete Modell der Basisfunktionalität des Fehlertoleranzmechanismus vor:

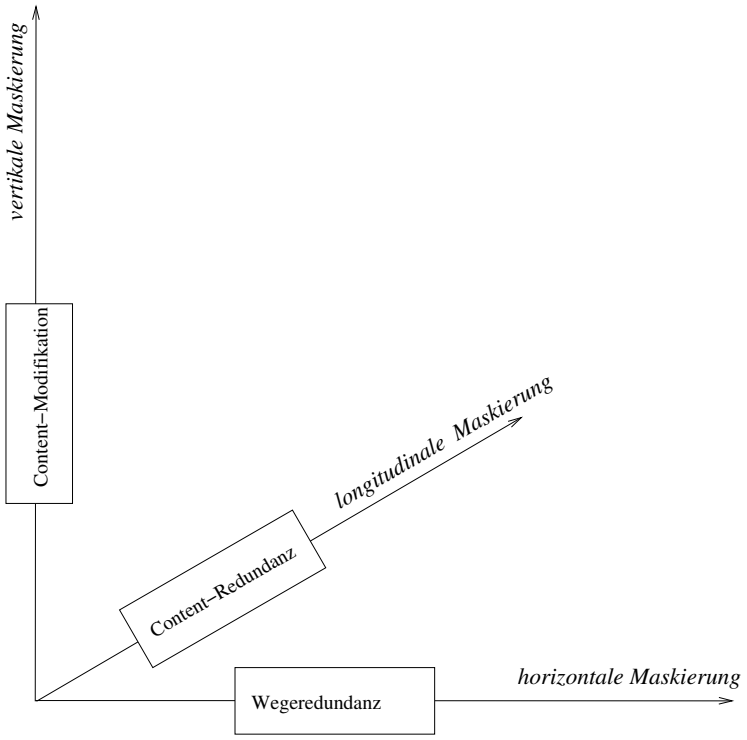

Abbildung 4.6: Funktionale Bausteine der adaptiven Fehlertoleranz

Aufbauend auf die präsentierten drei Kategorien von Applikationsfehlern führen wir folgende Mechanismen der *mehrdimensionalen Fehlerbehandlung* ein:

- **Longitudinale Maskierung**: Redundanter Content
- **Vertikale Maskierung**: Content-basierte Adaption
- **Horizontale Maskierung**: Redundante Pfade

Die Begriffe leiten wir aus der Sichtweise eines Endsystems auf netzbasierte, verteilte Applikationen ab.
Mit *longitudinaler Fehlermaskierung* bezeichnen wir die Nutzung von im Netz vorhandenen Kopien eines von einem Endsystem angefragten Content. Dem Endsystem bleibt dabei der tatsächliche Ursprung des Content im Netz verborgen. Ein einfaches Beispiel für einen bereits häufig genutzten derartigen Mechanismus sind die in Abschnitt 3.3.2 präsentierten Cacheing-Systeme. Ein Benutzer fragt ein bestimmtes Dokument aus dem Netz an. Ist dieses Dokument in einem Cache gespeichert, wird dieses an den Benutzer gesendet. Dem Benutzer bleibt verborgen, dass er nicht das Original-Dokument von einem spezifizierten Server bekommen hat, sondern eine (im Idealfall) identische, im Cache vorhandene, Kopie.

Vertikale Fehlermaskierung berücksichtigt ausschließlich die beteiligten Endsysteme. Hier kann man unterscheiden in:

- Content-modifizierende Verfahren
- An den Content angepasste, nicht-modifizierende Verfahren

Content-modifizierende Verfahren passen die übertragenen Daten an. Dies kann zum einen die klassische Transkodierung sein, die beschriebene Ressourcenmängel in Endsystemen maskiert und somit die Verarbeitung von Daten in Endsystemen ermöglicht. Zum anderen kann es auch notwendig sein, in Kooperation mit horizontaler Maskierung, Content speziell zu kodieren, z.B. durch Multiple Description Coding (siehe Abschnitt 3.5.2).
An den Content angepasste nicht-modifizierende, vertikale Fehlermaskierung greift in den Datentransport unter Berücksichtigung der Applikationsschicht ein. Ein Beispiel hierfür ist die Differenzierung von Paketen eines Datenstroms nach Prioritäten [cFLKP99].

Die *horizontale Fehlermaskierung* baut auf dem in Abschnitt 2.1.1 beschriebenen Prinzip der möglichen Pfadredundanz im Internet auf. Kommt es zu Ausfällen in Teilen des Internets, kann die Möglichkeit bestehen, über sequentiell verkettete Teilverbindungen die Konnektivität zwischen den Kommunikationspartnern auf einem zusammengesetzten, alternativen Pfad zu erhalten. Dieses Prinzip nutzen wir sowohl zum Schutz von Unicast-Verbindungen als auch für Anwendungen der Gruppenkommunikationen.

Aufbauend auf die drei Basisfunktionalitäten schlägt diese Arbeit vor, einen an die jeweils zu schützende Applikation adaptierten, Fehlertoleranzmechanismus bereitzustellen.
Für den Fall einer Client-Server-basierten Video on Demand-Applikation könnte dies z.B. eine Kombination aus horizontaler und vertikaler Fehlermaskierung sein, die das in Abschnitt 3.5.1 präsentierte PPD-TCP, transparent gegenüber den Endsystemen im Netz realisiert.
Zusätzlich zum Schutz gegenüber Ausfällen im Netz könnte die beschriebene Applikation auch noch gegen einen Ausfall der Serverarchitektur durch longitudinale Fehlermaskierung geschützt werden. Ein angefragtes Video müsste in diesem Fall von einer alternativen Stelle im Netz bezogen werden.

Im nächsten Abschnitt führen wir die adaptive Fehlertoleranz als Netzdienst ein. Die präsentierten Basiseigenschaften realisieren wir durch integrierte, kooperierende Overlay-Netze und geben einen Überblick über die Architektur des Dienstes.

4.2.3 Dienstarchitektur

Der Beitrag dieser Arbeit ist ein Fehlertoleranzmechanismus, der Applikationen direkt unterstützt. Wir schlagen vor, die in Abschnitt 4.2.2 dargelegten Basisfunktionen der adaptiven Fehlertoleranz an die jeweils zu schützenden Applikationen anzupassen und kombiniert als Netzdienst anzubieten. In diesem Abschnitt führen wir die Architektur dieses Netzdienstes ein.

Für die Bereitstellung der longitudinalen, vertikalen und horizontalen Fehlertoleranz als Netzdienst werden verschiedene, teilweise funktionsübergreifende Komponenten benötigt.

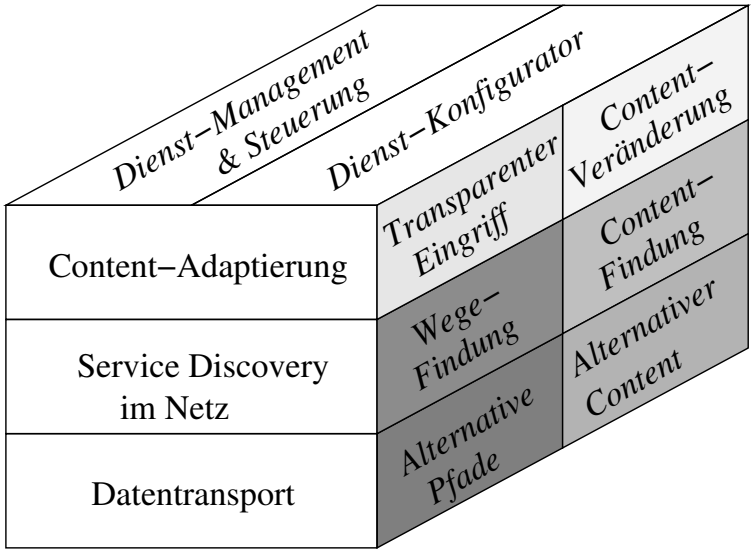

Abbildung 4.7: Komponenten des adaptiven Fehlertoleranzdienstes

In Abbildung 4.7 sind diese Komponenten des adaptiven Fehlertoleranzdienstes dargestellt. Der vorgeschlagene Netzdienst basiert auf drei Komponenten:

- Datentransport
- Service Discovery
- Content-Adaptierung

Die präsentierten Basisfunktionalitäten der adaptiven Fehlertoleranz werden in der Architektur des Dienstes abgebildet. Die longitudinale und auch die horizontale Fehlermaskierung benötigen Methoden um Daten innerhalb des Netzes transportieren zu können. Der *Datentransport* ist somit die grundlegende Komponente des Fehlertoleranzdienstes.
Mit *Service Discovery* bezeichnen wir die Komponente des Dienstes, die alternative Pfade im Netz bzw. auch alternativen Content im Netz findet und konfiguriert. Die Datentransportkomponente leitet die Daten im Anschluss durch das Netz. Somit sehen wir eine enge Verzahnung der beiden Komponenten.
Die Komponente der *Content-Adaptierung* unterstützt eine vertikale Fehlermaskierung. Hier ist einerseits eine zu den anderen Komponenten unabhängige Arbeitsweise vorstellbar. Die teilweise in Abschnitt 3.5 präsentierten (eindimensionalen) Fehlertoleranzmechanismen, die nur kommunizierende Endsysteme berücksichtigen, können so unterstützt werden. Andererseits ist auch eine Interaktion mit den anderen Komponenten des Fehlertoleranzdienstes denkbar. So können die in Abschnitt 3.5.2 präsentierten Kodierungen, die eine vertikale Fehlermaskierung bzw. Fehlerverschleierung darstellen, mit horizontaler oder auch longitudinaler Fehlermaskierung kombiniert werden. Eine Interaktion mit der Datentransport- und auch der Service Discovery-Komponente ist deshalb eine wichtige Voraussetzung.

Die vorgestellten Komponenten werden durch einen komponentenübergreifenden *Dienst-Konfigurator* kombiniert und an eine entsprechend zu schützende Applikation angepasst. Abhängig davon, ob es sich im Einzelfall um Unicast- oder Multicast-Kommunikation handelt, werden entsprechende alternative Datenpfade im Netz gesucht. Ist es möglich, und

gewünscht, die Applikation durch alternativen Content zu schützen, muss dieser bereitgestellt werden. Der Datentransportmechanismus wird an die entsprechenden Gegebenheiten angepasst. Soll zusätzlich eine vertikale Fehlermaskierung stattfinden, wird der Fehlertoleranzdienst ebenfalls durch den Dienst-Konfigurator angepasst, gegebenenfalls durch Interaktion mit den anderen Komponenten.
Des Weiteren benötigt der adaptive Fehlertoleranzdienst eine Dienststeuerung bzw. ein Dienst-Management. Wesentliche Funktionen sind hier das Starten, Stoppen, die Überwachung und auch die Steuerung des laufenden Fehlertoleranzdienstes. Das Dienst-Management benötigt interne Schnittstellen zu den Komponenten, um z.B. entsprechende Ersatzschaltverfahren einzuleiten. Außerdem sind externe Schnittstellen notwendig, um den Dienst von außen zu steuern. Dies kann z.B. durch einen Administrator aber auch durch die Endsysteme erfolgen.

Die beschriebene Trennung des Datentransports von der dazu gehörenden Signalisierung (Suche nach alternativen Pfaden bzw. alternativem Content) ist ein gängiges Prinzip in Kommunikationsnetzen.
Die in Abschnitt 3.3.1 angeführten fehlertoleranten Overlay-Netze bauen ebenfalls auf einer *logischen Trennung* von Datentransport und Signalisierung auf. Die Signalisierung erfolgt in den Architekturen outband, d.h. getrennt vom Datentransport. Gemeinsames Merkmal dieser fehlertoleranten Overlay-Netze ist jedoch die *topologische Abhängigkeit* der benützten Pfade im Netz für die Signalisierung und Transport. Die beschriebenen Architekturen trennen zwar logisch, benutzen aber *ein gemeinsames* Overlay-Netz für die Signalisierung und den Datentransport.
In dieser Arbeit schlagen wir vor, den Datentransport und die Signalisierung auch topologisch zu trennen. Auf die unterschiedlichen Anforderungen der Komponenten des adaptiven Fehlertoleranzdienstes gehen wir im nächsten Abschnitt ein.

4.2.4 Komposition durch integrierte Overlay-Netze

Wir realisieren die Komponenten der adaptiven Fehlertoleranz durch Overlay-Netze.
Die grundlegende Arbeit auf dem Gebiet der fehlertoleranten Overlay-Netze, *Resilient Overlay Networks* (siehe Abschnitt 3.3.1), führt an, dass eine alternative Route in einem Overlay-Netz für eine bestimmte Kommunikationsverbindung bereits durch einen einzigen Hop (siehe Abbildung 4.8) in ausreichender Qualität bereitgestellt werden kann.

Der in RON beschriebene Vorteil der Benutzung nur eines Hops liegt, gegenüber der Benutzung mehrerer Hops, darin, dass durch die Umleitung bedingte, zusätzliche Verzögerungen im Datenpfad klein gehalten werden können. Darüber hinaus ist kein komplexes Routing im Overlay-Netz erforderlich, somit wird eine zusätzliche mögliche Fehlerquelle eliminiert. Wie im RON-Projekt empirisch nachgewiesen, hat die Benutzung mehrerer, sequentiell geschalteter Hops im Overlay-Netz keinen Vorteil bei Ausfällen. Schon die Benutzung nur eines Hops für die Bereitstellung des alternativen Pfades umgeht Fehler im Netz in ausreichender Weise.
Daraus leiten wir die Anforderung an den Datentransport für den in dieser Arbeit präsentierten adaptiven Fehlertoleranzdienst ab:

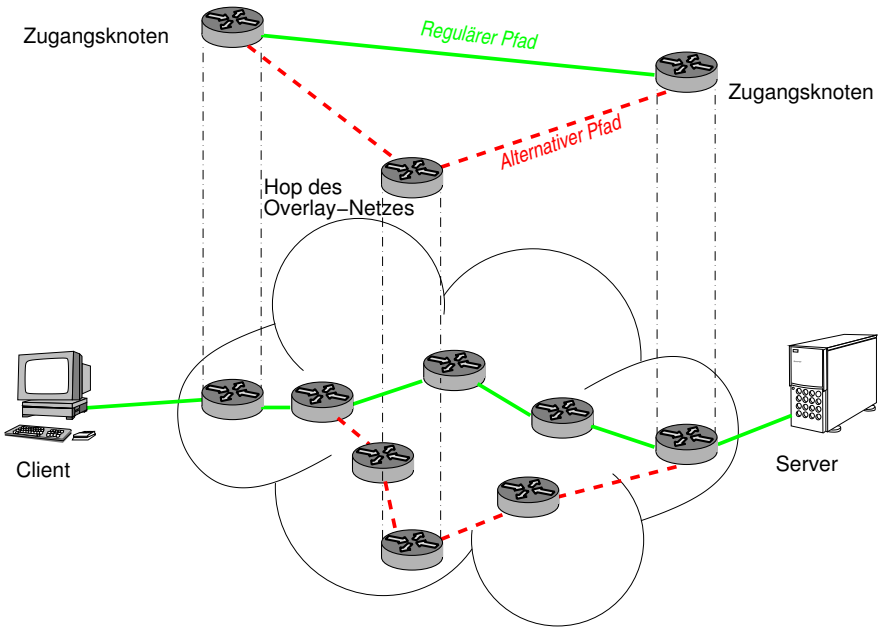

Abbildung 4.8: Auf nur einem Hop basierter, alternativer Pfad im Overlay-Netz

Der Datentransport auf dem alternativen Pfad soll in einem Overlay-Netz erfolgen, das nur auf einem einzigen Zwischenknoten (2 Hop) basiert.

In RON ist die Anzahl der Knoten aus denen ein Hop applikationsspezifisch ausgewählt wird, auf ca. 20 limitiert. Dies liegt in der mangelnden Skalierbarkeit der Signalisierung (Knotenfindung) begründet. Die Begrenzung der Anzahl der Hops zieht offensichtlich direkt eine Begrenzung der Zahl der Endsysteme nach sich, die den Dienst benützen können. Sämtliche Daten, die Endsysteme über das Netz austauschen, müssen über jeweils einen der Knoten geroutet werden.
Daraus leiten wir eine Anforderung an die Zahl der möglichen Knoten ab, aus denen im Einzelfall einer als Hop für ein Overlay-Netz benützt werden soll:

Die Zahl der Knoten bzw. Hops, aus denen *nur einer*, für den Transport von Daten einer bestimmten Applikation ausgewählt wird, soll nicht begrenzt sein.

Ist die Zahl der Knoten in einem fehlertoleranten Overlay-Netz nicht begrenzt, so ist prinzipiell die Voraussetzung gegeben, dass die Zahl der das Overlay-Netz benutzenden Endsysteme ebenfalls nicht begrenzt sein muss. Somit haben wir eine weitere Forderung an die adaptive Fehlertoleranz:

Die Zahl der Endsysteme, die den Fehlertoleranzdienst benützen können, soll ebenfalls nicht begrenzt sein.

Eine Vielzahl an zur Verfügung stehenden Knoten eines fehlertoleranten Overlay-Netzes, die für den Datentransport genutzt werden, reicht allein nicht aus, den gewünschten skalierbaren Netzdienst anbieten zu können. Eine Schlüsselfunktion liegt in der Auswahl eines entsprechenden Knotens, der für den Datentransport benutzt wird:

Einerseits wird durch die Auswahl eines Knotens die Güte des alternativen Weges festgelegt. Ist ein Knoten unpassend gewählt, so ist der alternative Pfad u.U. nicht disjunkt zum originalen Datenpfad, und würde bei einem Ausfall im originalen Datenpfad keinen Schutz bieten. Andererseits ist in diesem Zusammenhang auch die momentane Auslastung der einzelnen Knoten zu berücksichtigen. Ist ein Knoten überlastet, kann der alternative Pfad nicht zur Verfügung gestellt werden.
Hieraus leiten wir Anforderungen an die Wegefindung ab:

- Integration aller zur Verfügung stehenden Knoten des Datentransport-Overlay-Netzes.
- Ein Mechanismus, der die Güte des alternativen Pfades berücksichtigt.
- Ein Mechanismus, der die Auslastung der Knoten im Datentransport-Overlay-Netz mit einbezieht.

Die unterschiedlichen Anforderungen von Datentransport und Wegefindung bilden wir in topologisch getrennten Overlay-Netzen ab. In dieser Arbeit schlagen wir vor, den Datentransport in dynamischen Overlay-Netzen zur Verfügung zu stellen, angepasst an die jeweils zu schützende Applikation. Die Wegefindung stellt ebenfalls ein Overlay-Netz dar und konfiguriert das Datentransport-Overlay-Netz.
Dieses Grundprinzip findet sich auch in P2P-Netzen wie Gnutella [Kan01] wieder: Durch ein Overlay-Netz wird Content in einem auf Endsystemen aufbauenden, skalierbaren Netz gesucht. Diesen Suchvorgang kann man abstrahiert mit der Service Discovery-Komponente des Fehlertoleranzdienstes vergleichen. Ist der gesuchte Content auf einem entfernten Endsystem gefunden, wird er direkt an das anfragende Endsystem übermittelt. Der Datentransport erfolgt in Gnutella dann nicht innerhalb des P2P-Overlay-Netzes, sondern über eine direkte (TCP) Verbindung zwischen den Endsystemen. Abstrahiert betrachtet, ist diese direkte Verbindung mit der Datentransportkomponente des beschriebenen Fehlertoleranzdienstes vergleichbar.

Die beschriebene Ähnlichkeit zwischen der Wegefindung der adaptiven Fehlertoleranz und der Contentfindung in P2P-Netzen gilt nur von einem abstrahierten Niveau aus betrachtet. Im Detail unterscheiden sie sich:

- Bei der Contentfindung ist die Topologie des Overlay-Netzes nachrangig. Vorrangig wird ein bestimmter Content gesucht. Erst in Optimierungsverfahren, die auf eine effizientere Suche bzw. einen effizienteren Datenaustausch abzielen, wird die Topologie u.U. berücksichtigt.
- Die Wegefindung der adaptiven Fehlertoleranz hingegen sucht ausschließlich nach Knoten im Overlay-Netz, die gewisse topologische Anforderungen erfüllen.

Diesem Unterschied tragen wir in dieser Arbeit Rechnung, indem wir die Service Discovery-Komponente der adaptiven Fehlertoleranz ebenfalls durch unabhängige Overlay-Netze bereitstellen.
Somit werden die Komponenten Datentransport, Wegefindung und Contentfindung jeweils durch Overlay-Netze abgebildet. Diese Overlay-Netze kooperieren, sind aber topologisch voneinander getrennt. Das Dienst-Management integriert diese Overlay-Netze zu einem Netzdienst.

4.3 Gesamtmodell der fehlertoleranten Kommunikation

In diesem Abschnitt beschreiben wir die Einbettung des Fehlertoleranzdienstes in die Kommunikation. Wir gehen davon aus, dass spezifische, verteilte Applikationen durch den Fehlertoleranzdienst unterstützt werden. Prinzipiell unterscheiden wir in zwei Fälle:

- Transparenter Betrieb gegenüber Applikationen.
- Durch Applikationen gesteuerter Betrieb des Netzdienstes.

Das Konzept der transparenten Fehlertoleranz gegenüber Applikationen ist in der Telekommunikation seit langem üblich. Damit sind Mechanismen im Netz gemeint, die die Fehlertoleranz des Gesamtsystems erhöhen. Die Endgeräte, bzw. Applikationen auf den Endgeräten, bemerken diese Mechanismen aber nicht. Die einzig merkliche Veränderung aus Sicht der Endgeräte ist in diesem Fall eine Veränderung der Ausfallrate.
Die zweite betrachtete Betriebsweise des beschriebenen Fehlertoleranzdienstes ist die direkte Interaktion mit Applikationen bzw. Endsystemen. Dadurch ist es möglich den Fehlertoleranzdienst besser an die (u.U. sich dynamisch ändernden) Bedürfnisse der jeweiligen Applikationen anzupassen.

4.3.1 Transparente Fehlertoleranz

In diesem Abschnitt betrachten wir den transparenten Betrieb des beschriebenen Fehlertoleranzmechanismus. Ziel ist die Bereitstellung des Fehlertoleranzdienstes ohne dass es dafür Änderungen in den Endsystemen bedarf. Die notwendigen Änderungen werden im Netz vorgenommen, diese sind jedoch transparent gegenüber Applikationen bzw. Endsystemen. Dies unterscheidet die vorliegende Arbeit zu den in Abschnitt 3.3.1 beschriebenen (intransparenten) fehlertoleranten Overlay-Netze.
Unserer Ansicht nach ist die Transparenz des Fehlertoleranzmechanismus eine wichtige Voraussetzung um eine Nutzung in der Breite zügig zu ermöglichen. Kann der Dienst transparent betrieben werden, können alle bekannten Applikationen, ohne Modifikationen in den Endsystemen, diesen nach Bereitstellung sofort nutzen.
Sind Modifikationen in den Endsystemen notwendig, dann ist die Eintrittsbarriere für die Benutzung des Dienstes hoch: Die vorgeschlagenen intransparenten, fehlertoleranten Overlay-Netze führen proprietäre Schnittstellen ein, an die die Applikationen jeweils angepasst werden müssten.

Um eine transparente Nutzung des Fehlertoleranzdienstes zu ermöglichen, ist von Seiten des Netzdienstes eine Analyse und Kategorisierung des Datenverkehrs zwischen den jeweiligen Endsystemen notwendig. Der Dienst muss selbständig erkennen können, ob es sich z.B. um Unicast- oder Multicast-Datenverbindungen handelt.
Des Weiteren müssen auch Informationen aus höheren Schichten in die Kategorisierung des Datenverkehrs einfließen. Handelt es sich z.B. um eine auf dem HTTP-Protokoll basierte Client-Server-Anwendung, die ein bestimmtes Dokument aus dem Netz anfragt, dann ist es u.U. möglich dieses Dokument in Kopie von einem alternativen Server zu beziehen. Findet

die Kommunikation verschlüsselt statt, z.B. auf dem HTTPS-Protokoll, dann ist dies nicht möglich, da in diesem Fall zwar aus dem Datenstrom der angefragte Name des Servers extrahiert werden kann, nicht jedoch das angefragte Dokument.
Die Analyse der Daten muss also schichtenübergreifend erfolgen, um den Verkehr zwischen den Endsystemen zu charakterisieren.

Nach der Analyse und Kategorisierung des Datenverkehrs beginnt der Schutz des Datenverkehrs durch den beschriebenen Fehlertoleranzmechanismus. Da im transparenten Fall, keine Steuerung des Netzdienstes von Seiten der Endsysteme erfolgt, ist es notwendig, dass dem Fehlertoleranzdienst Anweisungen vorliegen, wie mit dem kategorisierten Datenverkehr zu verfahren ist. Die Betreiber des Fehlertoleranzdienstes geben eine *Policy* (Verfahrensweise) vor, wie kategorisierter Datenverkehr (abhängig vom jeweiligen Endsystem) behandelt wird. Im Rahmen dieser Arbeit gehen wir nicht auf mögliche Verfahrensweisen ein. Der beschriebene Fehlertoleranzdienst ist allgemein gehalten und kann von den Betreibern entsprechend konfiguriert werden. Eine Policy könnte z.B. sein: Den Datenverkehr generell durch horizontale Fehlermaskierung zu schützen. Führt dies nicht zum Erfolg, kann eine longitudinale und vertikale Fehlermaskierung hinzugezogen werden.

Der transparente Betrieb des Fehlertoleranzdienstes ähnelt abstrahiert dem Betrieb einer klassischen Firewall. Eine Firewall analysiert Datenverkehr und entscheidet dann, ebenfalls aufgrund von vorgegebenen Verfahrensweisen, wie mit dem Datenverkehr zu verfahren ist. Für die beteiligten Applikationen auf den Endsystemen ist dieser Mechanismus ebenfalls transparent.

4.3.2 Applikationsgesteuerte Fehlertoleranz

Der transparente Betrieb des Fehlertoleranzdienstes unterstützt Applikationen direkt, aber ohne deren Wissen bzw. ohne zusätzliche Interaktion mit den beteiligten Endsystemen.
Für die in Abschnitt 3.5 präsentierten Fehlertoleranzmechanismen in Endsystemen wäre es jedoch von Vorteil bzw. ist es teilweise Bedingung mit dem Netz zu kooperieren. Das beschriebene PPD-TCP und auch der MDC-Ansatz von Liang et al., setzen z.B. die Verfügbarkeit von disjunkten Pfaden im Internet voraus. Die Vorschläge gehen aber nicht darauf ein wie disjunkte Pfade im Internet tatsächlich zur Verfügung gestellt werden, sondern verweisen dabei allgemein auf fehlertolerante Overlay-Netze, wie z.B. RON.

Für die Maskierung von in Abschnitt 4.1.1 angeführten Fehlerzuständen in dynamischen bzw. statischen Ressourcen der Endsysteme ist ebenfalls eine Interaktion zwischen Endsystemen und dem vorgeschlagenen adaptiven Fehlertoleranzdienst notwendig. Derartige Fehlerzustände müssen vom Endsystem bzw. von den Applikationen festgestellt werden. Anschließend muss der Fehlertoleranzdienst vom Endsystem angepasst werden.

Für diese Art von Applikationen ist es somit notwendig eine externe Schnittstelle anzubieten, um den adaptiven Fehlertoleranzdienst vom Endsystem, bzw. durch die Applikation, einstellen zu können. Im Vergleich zur transparenten Fehlertoleranz stellt dies eine Vereinfachung des Ablaufs dar:
Im transparenten Fall wird Datenverkehr zunächst analysiert und anschließend eine Entscheidung getroffen, wie der Datenverkehr im Netzdienst behandelt wird. Danach wird der

Netzdienst durch den Dienst-Konfigurator entsprechend an die Applikation angepasst.
Für den applikationsgesteuerten Fall fällt die Analyse und Entscheidungsphase weg, der Netzdienst wird direkt durch die Applikation angepasst.

4.4 Komponenten des verteilten Fehlertoleranzdienstes

Die in 4.2.3 präsentierte allgemeine Architektur des adaptiven Fehlertoleranzdienstes wird in diesem Abschnitt detaillierter betrachtet.
Die präsentierten kooperierenden Overlay-Netze bauen auf im Internet verteilte Knoten auf. Im Folgenden gehen wir zunächst auf die Platzierung der einzelnen Knoten im Internet ein. Anschließend präsentieren wir den notwendigen modularen Aufbau der Knoten.

4.4.1 Platzierung der Knoten im Netz

Die in Abschnitt 2.2 präsentierten Overlay-Netze haben Gemeinsamkeiten im Aufbau. Die Overlay-Netze bauen auf einigen im Internet verteilten Netzelementen auf, die Knoten des Overlay-Netzes darstellen. Grundsätzlich gibt es zwei unterschiedliche Funktionsweisen der Knoten im Overlay-Netz:

- Knoten mit externer Schnittstelle, die den Zugang zum Overlay-Netz regeln.
- Knoten mit rein interner Funktionalität, die mit anderen Knoten des Overlay-Netzes Daten austauschen können.

Die in Abschnitt 4.2.4 detailliert präsentierten Anforderungen an die Architektur des mehrdimensionalen, adaptiven Fehlertoleranzdienstes bauen auf integrierten, kooperierenden Overlay-Netzen auf. Die einzelnen Overlay-Netze bestehen aus im Internet verteilten Knoten, die sich ebenfalls in diese Kategorien einteilen lassen.

Die in Abschnitt 4.3.1 geforderte mögliche Transparenz des Fehlertoleranzdienstes gegenüber Applikationen macht es notwendig, dass Knoten, die den Zugang zum Overlay-Netz bereitstellen, im direkten Pfad zwischen den kommunizierenden Endsystemen platziert sind. Durch die Platzierung muss sichergestellt sein, dass alle Datenpakete einer betreffenden, zu schützenden Applikation, einen Zugangsknoten passieren.
Bezugnehmend auf den in Abbildung 2.1 exemplarisch dargestellten Aufbau von IP-Netzen bzw. dem Internet, schlagen wir vor, die Zugangsknoten in Gateways bzw. Access-Routern, die wir in diesem Zusammenhang begrifflich als *Netzzwischensysteme* zusammenfassen, zu platzieren.
Die Platzierung in Core-Routern wäre aus zweierlei Gründen problematisch. Zum einen könnte ein Rerouting, während der Kommunikation zweier Endsysteme, den entsprechenden Datenstrom über einen anderen Pfad leiten. Somit würden entsprechende Datenpakete u.U. nicht mehr im Overlay-Netz aufgenommen werden. Zum anderen wäre es praktisch ein großer Aufwand, die beschriebene, notwendige Funktionalität für Zugangsknoten in Core-Routern zu integrieren.

Die Platzierung des Overlay-Netz-Zugangs direkt in den Endsystemen wäre technisch machbar, widerspräche aber dem Prinzip der Transparenz, da Änderungen in den Endsystemen anfallen würden.
Es verbleibt daher nur die Platzierung des Zugangs in Netzzwischensystemen. Im regulären Fall können Netzzwischensysteme nicht durch ein Rerouting umgangen werden. Die betreffenden Datenpakete müssen diese passieren. Im Vergleich zu Core-Routern ist der Aufwand der Realisierung auf diesen Systemen moderat, da das Verkehrsaufkommen in der Regel weit geringer ist.
In diesem Zusammenhang muss noch die tatsächliche Anbindung von Endsystemen an das Internet berücksichtigt werden. Sind Endsysteme z.B. in einem LAN über Multi-Homing in Kombination mit dem SCTP-Protokoll (siehe Abschnitte 3.2.2 und 3.5.1) angebunden, können während der Kommunikation unterschiedliche Gateways bzw. Access-Router benutzt werden. In letzterem Fall müssen alle möglichen Pfade der Endsysteme in das Internet berücksichtigt werden, und die beteiligten Netzzwischensysteme kooperieren. Eine Platzierung des Overlay-Netz-Zugangs auf den jeweiligen Gateways bzw. Access-Routern ist somit ebenfalls möglich.

Die Platzierung von Knoten mit rein interner Funktionalität im Overlay-Netz ist von der geforderten Transparenz unabhängig. Somit wäre prinzipiell eine Platzierung auf Core-Routern, Netzzwischensystemen oder auch dedizierten Endsystemen möglich.
Praktische Erwägungen führen aber auch in diesem Fall zu Einschränkungen. Die Platzierung in Core-Routern würde, aufgrund des in der Regel hohen Verkehrsaufkommens, einen erheblichen technischen Aufwand erfordern. Darüber hinaus können Änderungen im Core des Netzes, nur von den Netzbetreibern selbst vorgenommen werden, da nur diese Zugriff auf entsprechende Router haben. Der beschriebene Fehlertoleranzdienst wäre somit von der Bereitschaft der Netzbetreiber abhängig diesen einzuführen.
Der Zugriff auf Netzzwischensysteme kann aber, je nach Betreibermodell, unabhängig vom Netzbetreiber sein. In der Regel hat der Administrator eines LAN, Zugriff auf Netzzwischensysteme, die für die Anbindung an das Internet sorgen. Dies ist notwendig, um z.B. Sicherheitsfunktionen auf diesen Systemen zu installieren. Gleiches gilt für Endsysteme. Auch hier ist Unabhängigkeit vom Netzbetreiber gegebenen.
Die Platzierung von internen Knoten des Overlay-Netzes in den Endsystemen ist technisch gesehen die einfachste Lösung. Im Netz wären keinerlei Änderungen notwendig, es müssten nur zusätzliche Endsysteme an das Internet angeschlossen werden. Im Vergleich dazu wäre die Platzierung von internen Knoten auf Netzzwischensystemen sicherlich aufwendiger.
Berücksichtigt man aber die Tatsache, dass auf Netzzwischensystemen ohnehin Änderungen vorgenommen werden müssen, um den oben beschriebenen, transparenten Zugang zu ermöglichen, ist der zusätzliche Aufwand zur Realisierung eines internen Knotens auf diesen Systemen vernachlässigbar.

Im Rahmen dieser Arbeit schlagen wir in Kapitel 6 eine neue Architektur von Netzzwischensystemen vor, die mit dedizierten Endsystemen kooperieren. Somit können der transparente Zugang zum Overlay-Netz und auch die interne Funktionalität im Overlay-Netz, durch diese komponentenbasierten Netzzwischensysteme zur Verfügung gestellt werden. Der in dieser Arbeit beschriebene Fehlertoleranzdienst baut auf diesen Netzzwischensystemen auf.

4.4.2 Komponenten der Knoten

Die in Abschnitt 4.2.3 beschriebene Architektur des Fehlertoleranzdienstes bilden wir in diesem Abschnitt auf die Architektur der Knoten des Overlay-Netzes in Netzzwischensystemen ab. In Abbildung 4.9 sind die Komponenten eines Knotens gezeigt. Wesentlich ist

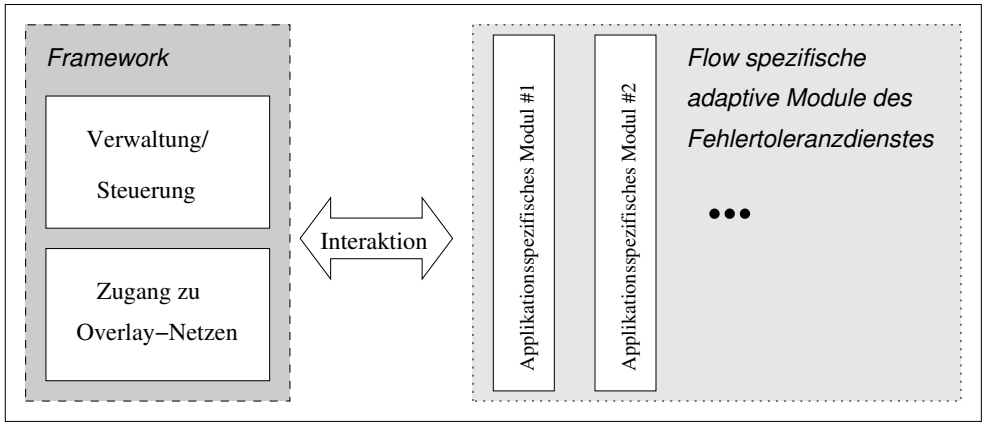

Abbildung 4.9: Komponenten eines Knotens

die Trennung in ein *Framework* und *flow- bzw. applikationsspezifische Module des Fehlertoleranzdienstes*. Das Framework fasst die allgemeinen, übergreifenden Funktionalitäten zusammen. So wird eine Verwaltungs- bzw. Steuerungseinheit benötigt, die den Knoten lokal steuert, überwacht und eine externe Schnittstelle zur Verfügung stellt. Des Weiteren regelt das Framework den Zugang des Knotens zu den in Abschnitt 4.2.4 beschriebenen Overlay-Netzen.

Die applikationsspezifischen Module übernehmen die Behandlung einzelner Applikationen. Sie werden durch das Framework konfiguriert und nutzen entsprechende Ressourcen, die von diesem zur Verfügung gestellt werden. Dies kann z.B. ein alternativer Pfad oder die Adresse einer Kopie eines angefragten Content sein, der bereits im Netz gefunden wurde. Ein Modul wird durch das Framework an die jeweils zu behandelnde Applikation angepasst, und kombiniert im Wesentlichen, die in Abschnitt 4.2.2 dargelegten Basisfunktionalitäten des Fehlertoleranzdienstes.

4.5 Modell der ladbaren adaptiven Fehlertoleranz

Der vorgeschlagene Fehlertoleranzdienst besteht aus kooperierenden Overlay-Netzen. Komponenten der Overlay-Netze sind über das Internet verteilte Knoten. In Abbildung 4.9 ist die Softwarearchitektur eines Knotens dargestellt.
In Abschnitt 4.2.4 sind die Anforderungen an die Overlay-Netze, bzw. Knoten, angeführt: Die Anzahl von am Fehlertoleranzdienst beteiligten Knoten soll möglichst nicht begrenzt sein. Um einen funktionierenden Fehlertoleranzdienst bereitzustellen, der diesen Anforderungen genügt, ist zum einen eine skalierbare Architektur notwendig. Auf diese gehen wir in Kapitel 5 ein. Zum anderen stellt die verteilte Software-Architektur Betreiber vor folgende Probleme beim Betrieb des Dienstes:

- Die praktische Bereitstellung des Dienstes bzw. der Knoten.
- Kontinuierliche Software-Updates, bzw. Adaption des Dienstes.

Eine mögliche Lösung dieser Probleme präsentieren wir in diesem Abschnitt. Wir schlagen vor, den adaptiven Fehlertoleranzdienst als ladbaren Netzdienst in Programmierbaren Netzen anzubieten. Im Folgenden gehen wir auf die Vorteile der Programmierbaren Netze für den Fehlertoleranzdienst ein und formulieren daraus resultierende Anforderungen an eine Architektur programmierbarer Knoten.

4.5.1 Fehlertoleranz als ladbarer Netzdienst

Die in Abschnitt 4.4.2 beschriebenen Knoten können mit einer klassischen Firewall verglichen werden. Die Platzierung der Knoten in Netzzwischensystemen entspricht in der Regel auch der Platzierung einer Firewall. Im transparenten Betrieb des Fehlertoleranzdienstes ähnelt der Ablauf auf den Knoten ebenfalls dem einer Firewall: Datenpakete, die den Knoten passieren, werden analysiert und entsprechend verarbeitet.

> **Eine prinzipielle Anforderung an die Knotenarchitektur ist es somit, in den Datenverkehr ab der Internetschicht eingreifen zu können.**

Des Weiteren müssen Funktionen zur Verfügung stehen, um den Zugang zu entsprechenden Overlay-Netzen bereitzustellen. Für den applikationsgesteuerten Betrieb ist es notwendig eine entsprechende Schnittstelle zur Verfügung zu stellen, um die Steuerung des Knotens seitens der Endsysteme zu ermöglichen. Die vertikale Fehlermaskierung bedingt die Ausführung u.U. komplexer Rechenprozesse (wie z.B. eine MPEG Transkodierung) auf einem Knoten.
Der Zugang zu Overlay-Netzen, sowie die externe Schnittstelle zu Endsystemen, als auch beschriebene Prozesse der vertikalen Fehlermaskierung können praktisch nur in der *Applikationsschicht* eines Knotens zur Verfügung gestellt werden.
Berücksichtigt man die in Abbildung 2.4 dargestellte Einbettung der einzelnen Schichten in ein Betriebssystem, stellt man fest, dass die den Knoten passierenden, zu behandelnden Datenpakete ausschließlich durch den Kern des Betriebssystem wandern. Der überwiegende Teil des Fehlertoleranzdienstes muss aber in der Applikationsschicht eines Knotens angesiedelt sein. Daraus folgern wir eine weitere Anforderung an die Bereitstellung des beschriebenen Fehlertoleranzdienstes:

> **Die Knotenarchitektur muss schichtenübergreifend sein. Die Funktionalitäten müssen sowohl im Kern des Betriebssystems als auch in der Applikationsschicht realisiert werden.**

Zusätzlich zu diesen statischen Eigenschaften eines Knotens des Fehlertoleranzdienstes kommen notwendige, dynamische Eigenschaften hinzu. Die präsentierten Module der Knotenarchitektur sind teilweise stark abhängig von den zu schützenden Applikationen. Soll eine neuartige Applikation durch den Fehlertoleranzdienst geschützt werden, muss diese dem Monitoring-Modul bekannt sein. Die Bereitstellung eines alternativen Contents ist ebenfalls, wie die Content-Adaption, abhängig von der jeweiligen Applikation. Auch die Suche nach alternativen Pfaden ist teilweise von der zu schützenden Applikation abhängig.
Die quasi kontinuierlich neu in Betrieb genommenen Applikationen im Internet, stellen an

den präsentierten Fehlertoleranzdienst somit u.U. ständig neue Anforderungen. Auf Seiten des Fehlertoleranzdienstes muss diesem Problem mit entsprechender Flexibilität begegnet werden können.
Dem gegenüber stehen Module, die von den zu schützenden Applikationen unabhängig sind, wie z.B. der Datentransportmechanismus oder die Dienststeuerung. Aber auch in diesem Fall ist im praktischen Betrieb des Dienstes mit Software-Updates zu rechnen. Somit ergibt sich eine weitere Anforderung an den Fehlertoleranzdienst bzw. die Knotenarchitektur:

> **Die Software der Knotenarchitektur muss an Anforderungen neuartiger Applikationen flexibel anpassbar sein.**

Die statischen und dynamischen Anforderungen an die Softwarearchitektur der Knoten, in Kombination mit den in Abschnitt 4.2.4 präsentierten, übergeordneten Anforderungen an den Fehlertoleranzdienst, stellen Betreiber des Fehlertoleranzdienstes vor folgendes praktisches Problem:

> **Auf einer angestrebten, unbegrenzten Zahl an beteiligten Knoten des Fehlertoleranzdienstes muss eine schichtenübergreifende Softwarearchitektur flexibel und schnell an neue Anforderungen anpassbar sein.**

Darüber hinaus befinden sich die entsprechenden Knoten in unterschiedlichen administrativen Domänen über das Internet verteilt. Eine Vorgehensweise des Betriebs könnte sein, dass die jeweiligen Administratoren der Domänen, quasi konstant, neue Versionen der Software des Fehlertoleranzdienstes auf die entsprechenden Knoten aufspielen. Aufgrund der angestrebten großen Zahl an beteiligten Domänen sehen wir aber praktische Hindernisse: Dieses Vorgehen wäre nicht automatisierbar, und somit würde der Faktor Mensch eine entscheidende Rolle für den (zuverlässigen) Betrieb des Fehlertoleranzdienstes einnehmen.

Im Rahmen dieser Arbeit schlagen wir vor, die adaptive Fehlertoleranz als ladbaren Netzdienst in Programmierbaren Netzen anzubieten. Der Vorteil dieses Vorgehens liegt klar in der Automatisierung der notwendigen Flexibilität des Fehlertoleranzdienstes.
Die in Abschnitt 2.3 dargestellten Programmierbaren Netze erfüllen zum einen die an die Knotenarchitektur gestellten statischen Anforderungen:

1. Auf den Datenverkehr kann ab der Internetschicht eingegriffen werden.
2. Schichtenübergreifende Funktionalitäten können in der Applikationsschicht bereitgestellt werden.

Die gegebene Möglichkeit, Software dynamisch auf programmierbare Knoten zu laden und diese auszuführen, erfüllt zum anderen die dynamischen Anforderungen an die Softwarearchitektur. Somit kann mittels der Programmierbaren Netze die notwendige Flexibilität zur Verfügung gestellt werden.

4.5.2 Anforderungen an programmierbare Knoten

Diese Arbeit präsentiert einen Fehlertoleranzdienst im Internet, der über programmierbare Knoten zur Verfügung gestellt wird. Die Softwarearchitektur auf den Knoten lässt sich somit in zwei Komponenten unterteilen: In die programmierbare Plattform zum einen und

den ladbaren Fehlertoleranzdienst zum anderen.
In diesem Zusammenhang ergeben sich außerdem zwei wesentliche Anforderungen an die programmierbare Plattform:

- Fehlertoleranz bzw. Zuverlässigkeit der Plattform
- Lokale Skalierbarkeit der Plattform

Zunächst soll der Fehlertoleranzdienst selbst zuverlässig zur Verfügung gestellt werden. Die programmierbare Plattform ist die Basis für die Ausführung des Fehlertoleranzdienstes. Tritt ein Versagen in der programmierbaren Plattform auf, dann können darauf laufende Dienste u.U. ebenfalls nicht mehr korrekt angeboten werden. Somit ist die Fehlertoleranz der programmierbaren Plattform eine Voraussetzung für eine mögliche Zuverlässigkeit der darauf laufenden Dienste.
Eine weitere Forderung liegt in der lokalen Skalierbarkeit einer programmierbaren Knotenarchitektur. Die in Abbildung 4.9 dargestellte komponentenbasierte Architektur eines Knotens startet applikationsspezifische Module für jede Applikation, die den Fehlertoleranzdienst benutzt. Diese u.U. rechenintensiven Prozesse müssen von der programmierbaren Plattform in skalierbarer Weise unterstützt werden können.

Die in Abschnitt 3.7 dargestellten Architekturen programmierbarer Knoten unterstützen die lokale Skalierbarkeit nur unzureichend. Auch die Zuverlässigkeit der vorgeschlagenen programmierbaren Knoten wird ebenfalls nur am Rande betrachtet.

Aus diesem Grund präsentieren wir in Kapitel 6 eine neuartige, skalierbare, fehlertolerante programmierbare Knotenarchitektur. Dadurch teilen wir unseren Ansatz in: *Fehlertoleranz als Netzdienst*, und *fehlertolerante Bereitstellung des Netzdienstes* auf.
Der Fehlertoleranzdienst wird als ladbarer bzw. programmierbarer Dienst auf einer Plattform angeboten. Die Bereitstellung des Dienstes erfolgt durch eine fehlertolerante programmierbare Plattform. Somit bildet das Zusammenspiel einen **fehlertoleranten Fehlertoleranzdienst**.

4.6 Zusammenfassung

Dieses Kapitel gibt einen Überblick über den Beitrag dieser Arbeit. Zunächst werden die zugrundeliegenden Systemmodelle von über das Internet kommunizierenden, verteilten Applikationen präsentiert, und darauf die in dieser Arbeit behandelten Fehler modelliert. Zur Behandlung der Fehler führen wir die mehrdimensionale, adaptive Fehlertoleranz als Netzdienst ein. Die Basisfunktionalität der adaptiven Fehlertoleranz besteht aus horizontaler, vertikaler und longitudinaler Fehlermaskierung bzw. Fehlerabschwächung. Wir schlagen vor, den Netzdienst durch Overlay-Netze in über das Internet verteilten Netzzwischensystemen bereitzustellen. Auf die Basisfunktionalitäten zugeschnittene kooperierende Overlay-Netze sind im vorgeschlagenen Fehlertoleranzdienst integriert.
Des Weiteren gehen wir auf einen möglichen transparenten und applikationsgesteuerten Betrieb des Netzdienstes ein. Das Zusammenspiel des Fehlertoleranzdienstes mit den zu schützenden, auf Endsystemen basierten, verteilten Applikationen wird dargestellt.
Zum Schluss erläutern wir noch die Bereitstellung des Dienstes in Netzzwischensystemen. Wir schlagen vor, den Fehlertoleranzdienst auf dedizierten, programmierbaren Knoten zur Verfügung zu stellen. Somit präsentieren wir einen fehlertoleranten Fehlertoleranzdienst bestehend aus:

- Adaptivem, ladbaren Fehlertoleranzdienst
- Skalierbarer, fehlertoleranter und programmierbarer Basisplattform für den Fehlertoleranzdienst

In den folgenden Kapiteln gehen wir auf den ladbaren Fehlertoleranzdienst und die programmierbare Plattform im Detail ein.

Kapitel 5

Ein ladbarer Fehlertoleranzdienst in Netzzwischensystemen

In diesem Kapitel präsentieren wir die Architektur des ladbaren Fehlertoleranzdienstes. Zunächst geben wir einen Überblick über die einzelnen Komponenten. Im Folgenden gehen wir detailliert auf die Bereitstellung und Anbindung des Dienstes an Overlay-Netze ein. Die eingeführten kooperierenden Overlay-Netze sind ein Hauptbeitrag dieser Arbeit.
Des Weiteren stellen wir einen neuen Datentransportmechanismus in Overlay-Netzen vor. Dieser vorgeschlagene Mechanismus erlaubt effizienten, gegenüber Endsystemen transparenten Transport von Nutzdaten durch den Fehlertoleranzdienst.

5.1 Überblick der Dienstarchitektur

In diesem Abschnitt geben wir einen Überblick über die Softwarearchitektur des ladbaren Fehlertoleranzdienstes. Zunächst präsentieren wir den modularen Aufbau, und stellen die in den Abschnitten 4.2.3 und 4.4.2 beschriebene Architektur des Fehlertoleranzdienstes im Detail dar.
Außerdem gehen wir auf das Zusammenspiel der zwei wesentlichen funktionalen Komponenten des Fehlertoleranzdienstes ein: Die integrierten, kooperierenden Overlay-Netze und die applikationsspezifischen Module.

5.1.1 Modularer Aufbau

In Abbildung 5.1 ist der modulare Aufbau der Softwarearchitektur des Fehlertoleranzdienstes auf einem Zugangsknoten dargestellt. Zunächst wird in zwei Komponenten unterteilt: In ein *Framework* und in *flow- bzw. applikationsspezifische Module des Fehlertoleranzdienstes.* Das Framework fasst die allgemeinen, übergreifenden Funktionalitäten zusammen.
So wird eine Verwaltungs- bzw. Steuerungseinheit benötigt, die den Knoten lokal steuert, überwacht und auch eine externe Schnittstelle zur Verfügung stellt. Das Dienst-Management koordiniert alle lokalen Funktionalitäten und kooperiert mit anderen beteiligten Knoten. Ein

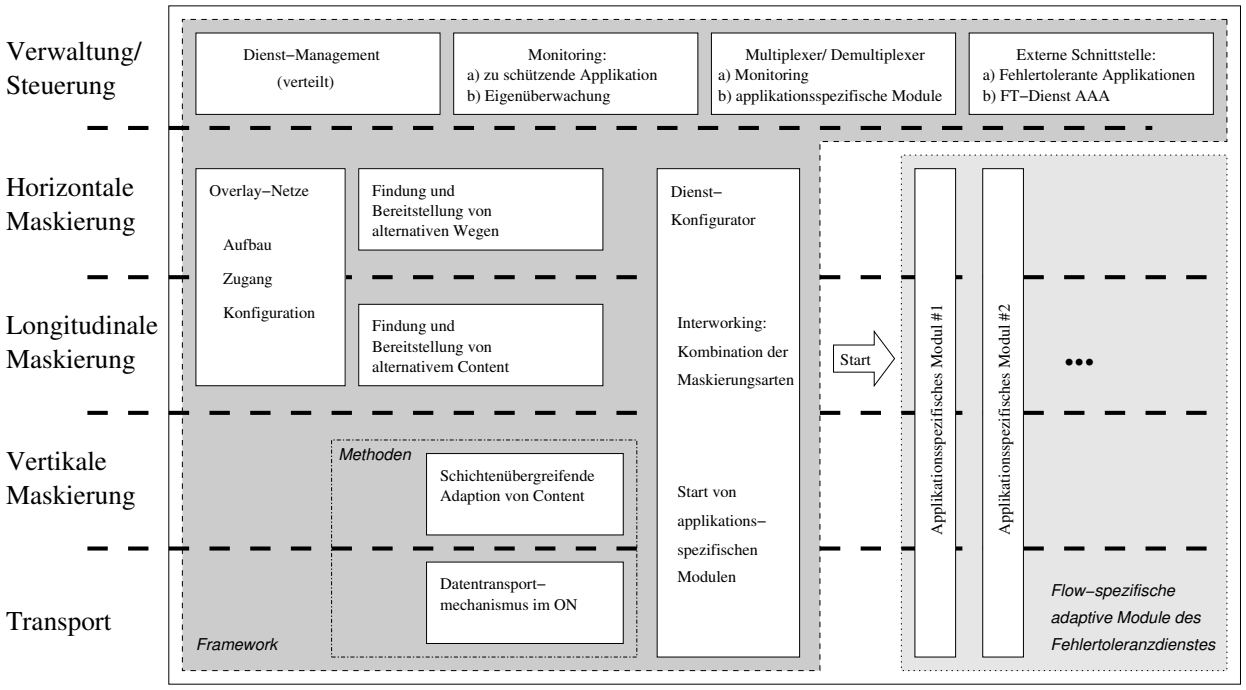

Abbildung 5.1: Software-Module des Fehlertoleranzdienstes

wesentlicher Teil der Steuerung ist auch die Überwachung. Der Knoten muss lokal überwacht werden. Am Dienst beteiligte, entfernte Knoten werden ebenfalls überwacht. Für den transparenten Betrieb des Fehlertoleranzdienstes ist es auch erforderlich den durch den Zugangsknoten durchfließenden Verkehr zu überwachen, ähnlich einer Firewall. Der vorgeschlagene Fehlertoleranzdienst unterscheidet durch den Knoten durchfließenden Verkehr nach Instanzen von Applikationen (engl. *sessions*). Zur differenzierten Verarbeitung wird der Verkehr session-basiert durch einen Demultiplexer im Framework aufgeteilt. Nach der paketbasierten Verarbeitung werden Datenpakete über einen Multiplexer weiterverschickt. Zudem stellt die Steuerungseinheit eines Zugangsknoten noch eine externe Schnittstelle zur Verfügung.
Für die horizontale und longitudinale Fehlermaskierung ist es notwendig, dass der Knoten Teilnehmer in Overlay-Netzen ist. Der Aufbau und Zugang zu den Netzen wird ebenfalls durch das Framework zur Verfügung gestellt. Darauf gehen wir in den Abschnitten 5.1.2 und 5.2 näher ein. Außerdem beinhaltet das Framework auch Methoden der vertikalen Fehlermaskierung und des Datentransports.

Im transparenten Betrieb des Zugangsknoten werden die durch den Knoten laufenden Datenpakete von der Monitoring- bzw. Überwachungskomponente analysiert. Soll eine entsprechend erkannte Applikation geschützt werden, initiiert dies der Dienst-Konfigurator: Je nach Applikation wird aus den entsprechenden Bausteinen der Fehlermaskierung ein applikationsspezifisches Modul maßgeschneidert, das anschließend die Behandlung der entsprechenden Instanz einer Applikation übernimmt.
Für den applikationsgesteuerten Fall entfällt das Monitoring des Datenverkehrs. Über eine externe Schnittstelle wird ebenfalls ein maßgeschneidertes, applikationsspezifisches Modul gestartet. Mit diesem Modul kommuniziert die anfordernde Applikation auf den entsprechenden Endsystemen dann direkt.

Die applikationsspezifischen Module sind an die jeweilige Instanz der zu schützenden Applikation adaptiert. Genauer gehen wir darauf in Abschnitt 5.1.3 ein.

Die in Abschnitt 4.4.1 angeführten Knoten mit rein interner Funktionalität weisen eine ähnliche Architektur zu den Zugangsknoten auf.
Auch hier gibt es eine Unterteilung in zwei Komponenten: Framework und applikationsspezifische Module. Im Framework gibt es ebenfalls die einzelnen Module, die den Zugang zu entsprechenden Overlay-Netzen bereitstellen. Im Unterschied zu den Zugangsknoten erfolgt die Konfiguration und der Start der applikationsspezifischen Module ausschließlich über eine externe Schnittstelle. Ein Dienst-Konfigurator ist somit nicht zwingend erforderlich für rein interne Funktionalität.
Die große Ähnlichkeit zwischen Zugangsknoten und rein internen Knoten wird im vorgeschlagenen Fehlertoleranzdienst genutzt, indem Zugangsknoten zusätzlich auch auch als Knoten im Sinne interner Funktionalität betrachtet werden.

5.1.2 Kooperierende Overlay-Netze im Framework

Die in Abschnitt 4.2.4 vorgeschlagenen, kooperierenden und integrierten Overlay-Netze können direkt auf die Komponenten bzw. Module der Knoten abgebildet werden:

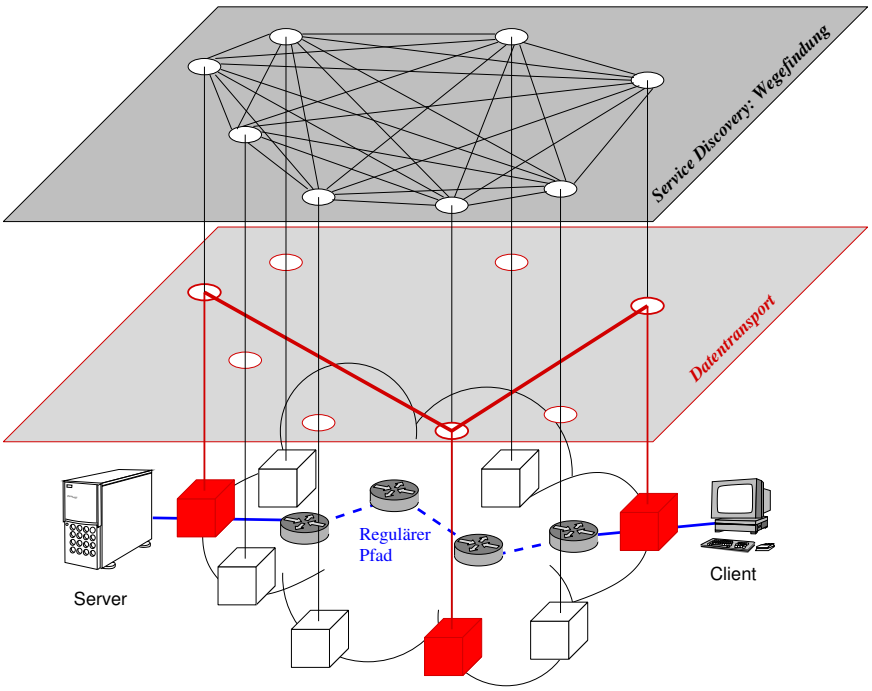

Abbildung 5.2: Exemplarisches Zusammenspiel des Wegefindung-Overlay-Netzes mit dem dynamischen Overlay-Netz des Datentransports

Die vorgeschlagenen Overlay-Netze zur Service-Discovery werden durch die Module der alternativen Wegefindung und der alternativen Contentfindung im Framework bereitgestellt. Diese Overlay-Netze müssen sich zwar einer sich ändernden Topologie im Netz anpassen können, haben aber statischen Charakter.

Die konfigurierten, applikationsspezifischen Module stellen dynamische, minimale Overlay-Netze dar, die den Datentransport im Einzelfall organisieren, und gegebenenfalls entsprechenden Content applikationsspezifisch anpassen.
In Abbildung 5.2 ist dies exemplarisch dargestellt. Client und Server sind über das Internet miteinander verbunden. Server- und Client-Applikationen tauschen auf dem regulären Datenpfad Daten durch das Netz aus. Das Overlay-Netz zur Findung alternativer Wege konfiguriert für diese Applikation dynamisch ein minimales Overlay-Netz, das den Datentransport auf einem gefundenen, alternativen Pfad organisiert. Der Übersicht halber wird in diesem Beispiel nur die horizontale Maskierung berücksichtigt.

5.1.3 Applikationsspezifische Module

Die applikationsspezifischen Module übernehmen die Behandlung einzelner Applikationen. Sie werden durch das Framework konfiguriert und nutzen entsprechende Ressourcen, die dieses zur Verfügung stellt. Dies kann z.B. ein alternativer Pfad oder die Adresse einer Kopie eines angefragten Content sein, der bereits im Netz gefunden wurde. Ein applikationsspezifisches Modul des Fehlertoleranzdienstes kombiniert im Wesentlichen folgende Aufgaben:

- Ersatzschalten
- Bereitstellung redundanten Contents
- Content-Adaptierung
- Kommunikation mit weiteren verteilten applikationsspezifischen Modulen

Je nach Anforderung stellt ein Modul im Zugangsknoten verschiedene Verfahren des Ersatzschaltens bereit. Des Weiteren ist es die Aufgabe eines Moduls, alternativen Content von, vom Framework gefundenen Knoten im Netz, zu übertragen und lokal bereitzustellen. Wesentlich in diesem Zusammenhang ist die Kommunikation mit anderen, im Netz verteilten, applikationsspezifischen Modulen.

Um vertikale Fehlermaskierung zu ermöglichen, benötigen die Module zusätzliche Methoden, um Content entsprechend an die Anforderungen von Endsystemen zu adaptieren.
Von Seiten des Frameworks werden in diesem Fall allgemeine Methoden zur Verfügung gestellt. Dies kann z.B. eine Transkodierung zwischen standardisierten Datenformaten sein. Im applikationsgesteuerten Betrieb ist seitens der Endsysteme damit lediglich eine Signalisierung an den Fehlertoleranzdienst notwendig, die eine entsprechende Transkodierung anfragt. Tritt bei den Endsystemen bzw. Applikationen der Wunsch auf, Teile der Applikation auszulagern, weil z.B. die in Abschnitt 4.1.1 beschriebenen dynamischen oder statischen Ressourcen eines Endsystems nicht genügen, wird dedizierter Code als Teil des applikationsspezifischen Moduls ausgeführt. Um diesen Ansatz offen und flexibel zu gestalten, muss es möglich sein, spezifischen Code (u.U. von dritter Stelle im Netz) in das Modul zu laden und es somit entsprechend anzupassen. Die Technologie der Programmierbaren Netze (siehe Abschnitt 2.3) unterstützt das dynamische Laden bzw. Nachladen von Code, in Kombination mit entsprechend gefordertem Zugriff auf den Datenverkehr. Dies ist einer der Gründe für die Wahl der Programmierbare Netze als Grundlage für den Fehlertoleranzdienst.

Die angesprochene Verlagerung von Rechenprozessen zeigt auch eine gewisse Ähnlichkeit zu Aspekten des Grid-Computing (siehe Abschnitt 2.2.3). Auch deshalb lehnen wir die, in Kapitel 6 als Basis für den Fehlertoleranzdienst, eingeführte programmierbare Plattform an vorgeschlagene Architekturen des Grid-Computing an.

5.1.4 Dynamik des Dienstes

Die im vorhergehenden Abschnitt angesprochene notwendige Flexibilität bzw. Dynamik des Fehlertoleranzdienstes hat im Wesentlichen zwei Aspekte:

- Dynamische Anpassung innerhalb des Dienstes.
- Flexible Anpassung des Frameworks.

Die dienstinterne Dynamik beschränkt sich auf die applikationsspezifischen Module. Durch das Framework, oder auch über eine externe Schnittstelle, werden Module an zu schützende Applikationen angepasst und bereitgestellt. Der in [FHSZ02] vorgeschlagene Aufbau eines Netzdienstes aus Teilmodulen durch dynamisches Linken, ist Grundlage für die Komposition von, an Applikationen angepasste Module des Fehlertoleranzdienstes.
Innerhalb des Frameworks müssen somit Funktionalitäten zur Verfügung stehen, die dynamisch applikationsspezifische Module bereitstellen können. Für den Fall des transparenten Betriebs übernimmt der Dienst-Konfigurator diese Funktion. Bei der applikationsgesteuerten Fehlertoleranz wird diese Funktionalität über die externe Schnittstelle im Framework bereitgestellt.
In beiden Fällen ist es notwendig, die beschriebenen Teilmodule dem Fehlertoleranzdienst zur Verfügung zu stellen. Die in Abschnitt 3.7.1 präsentierten Architekturen von Programmierbaren Netzen schlagen vor, Teilmodule eines Dienstes auf im Netz befindlichen Servern (engl. *Repositories*) zur platzieren. Von diesen Servern können Teilmodule dann auf programmierbare Knoten geladen werden und stehen somit dem Fehlertoleranzdienst zur Verfügung. Diese grundlegende Funktionalität von Programmierbaren Netzen ist durch die in Kapitel 6 vorgeschlagene Architektur abgedeckt.

Die applikationsspezifischen Module stellen aufgrund sich wandelnder Applikationen eine hohe Anforderung an die Flexibilität. Betrachtet man isoliert das Framework des Fehlertoleranzdienstes, stellt man ebenfalls ein gewisses Maß an notwendiger Dynamik fest: Für den transparenten Betrieb müssen dem Monitoring-Modul des Frameworks entsprechende, neue Applikationen bekannt sein; somit ist eine häufig notwendige Anpassung zu erwarten.
Der überwiegende Teil des Frameworks hat hingegen eher statischen Charakter. Die Funktionalität der Anbindung an Overlay-Netze wird sich z.B. nicht so häufig ändern, wie dies bei den dienstspezifischen Modulen der Fall ist. Lediglich gelegentliche Updates der Software sind zu erwarten. Durch Programmierbare Netze hat ein Administrator eines programmierbaren Knotens eine effiziente Möglichkeit, das System auf dem neuestem Stand zu halten.

Die in Abschnitt 4.5.1 dargelegten, allgemeinen Anforderungen – in Kombination mit den konkretisierten, dynamischen Anforderungen – legen es somit nahe den Fehlertoleranzdienst in Programmierbaren Netzen anzubieten.

5.2 Peer-to-Peer-basierte Basis-Signalisierung

In diesem Abschnitt stellen wir im Detail die Anbindung der einzelnen beteiligten Knoten in den Fehlertoleranzdienst dar. Im Vergleich zu den in Abschnitt 3.3.1 präsentierten Ansätzen, schlagen wir in dieser Arbeit vor, den Fehlertoleranzdienst aus mehreren kooperierenden Overlay-Netzen aufzubauen.
Im Folgenden beschreiben wir ein skalierbares, strukturiertes Overlay-Netz, dessen Ziel es ist, alle beteiligten Knoten in den Fehlertoleranzdienst zu integrieren. Dieses Overlay-Netz bildet die Basis für die in den Abschnitten 5.3 und 5.4 vorgestellten dedizierten Overlay-Netze zur Wege- und Contentfindung.

5.2.1 Übergeordnetes durchgängiges P2P-Netz

In Abbildung 5.3 ist die Architektur des übergeordneten, strukturierten P2P-Overlay-Netzes dargestellt. Aufgabe dieses Overlay-Netzes ist die Integration aller im Internet verteilten Knoten, die an der Bereitstellung des Fehlertoleranzdienstes beteiligt sind.

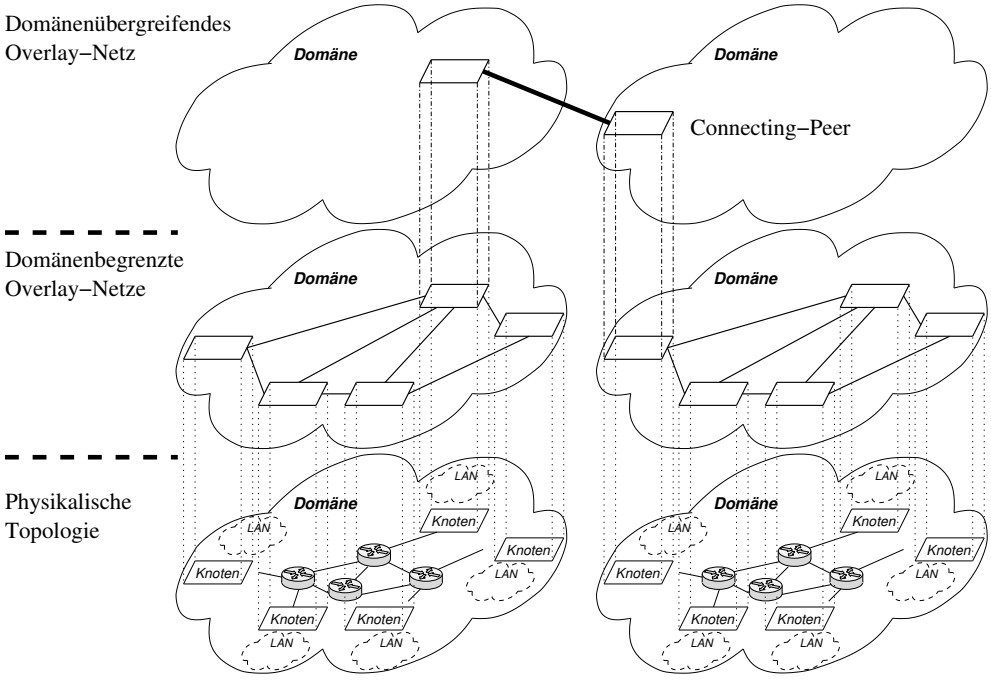

Abbildung 5.3: Ausschnitt des übergeordneten P2P-Overlay-Netzes

Wir schlagen eine Strukturierung in zwei Hierarchieebenen vor. Die niedere Ebene umfasst Overlay-Netze, bestehend aus Knoten, die jeweils durch administrative Domänen (wie z.B. ein Autonomes System, bzw. auch Teile eines Autonomen Systems) begrenzt sind. Innerhalb der niederen Ebene agieren die beteiligten Knoten als *Peers* in einem P2P-Overlay-Netz.
Die Bildung dieser domänenbegrenzten P2P-Netze geschieht ohne zentrale Instanz und erfolgt durch Selbstorganisation [Bac03]. Die Umgehung einer zentralen Instanz ist Grundprinzip der gesamten Dienstarchitektur. Dadurch werden mögliche *Single-Points-of-Failure* vermieden.

Im Unterschied zu den in Abschnitt 2.2.4 angeführten P2P-Netzen, besteht unser Vorschlag ausschließlich aus Peers, die in Netzzwischensystemen alloziert sind.
Innerhalb des domänenbegrenzten, niederschichtigen P2P-Netzes wählen die Peers jeweils einen *Connecting-Peer*. Dieser Connecting-Peer ist dann zum einen Teilnehmer im niederen P2P-Netz seiner Domäne; und zum anderen gleichzeitig ein Peer im höherschichtigen P2P-Netz. Das höherschichtige P2P-Netz besteht aus Connecting-Peers aller am Fehlertoleranzdienst beteiligten administrativen Domänen. Somit ist das höherschichtige P2P-Netz domänen- bzw. netzübergreifend aufgebaut. Für das höherschichtige P2P-Netz schlagen wir einen Netzaufbau vor, der vom Gnutella P2P-Netz [gnu] adaptiert ist.

Das übergeordnete P2P-Netz lehnt sich an den Aufbau des Internets an. Jede am Fehlertoleranzdienst beteiligte administrative Domäne ist durch einen *Connecting-Peer* in der höheren Schicht des P2P-Netzes vertreten. Nimmt man eine weite Verbreitung und Akzeptanz des Fehlertoleranzdienstes im Internet an, entspricht die Anzahl der zu erwartenden Connecting-Peers in etwa der im Internet konnektierten Autonomen Systeme bzw. administrativen Domänen. In der höheren Schicht der P2P-Architektur ist somit gegenwärtig eine Größenordnung von ca. 10.000 Teilnehmern zu erwarten.
Im Gegensatz zu herkömmlichen P2P-Netzen, spielen zwei Punkte hier nur eine untergeordnete Rolle:

- Bandbreitenproblematik
- Separation aufgrund der Netzdynamik

Das übergeordnete P2P-Netz dient ausschließlich dazu, alle am Fehlertoleranzdienst beteiligten Knoten einzubinden und dedizierte Overlay-Netze für die Wege- (siehe Abschnitt 5.3) und Contentfindung (siehe Abschnitt 5.4.3) aufzubauen. Somit werden über das übergeordnete P2P-Netz, in Relation zu den P2P-Netzen in Abschnitt 2.2.4, nur wenige Signalisierungsdaten versendet. Des Weiteren ist die Zahl der teilnehmenden Peers durch die Anzahl der beteiligten administrativen Domänen begrenzt. Das in herkömmlichen, unstrukturierten P2P-Netzen auftretende Problem, der u.U. nicht skalierenden Signalisierung durch begrenzt zur Verfügung stehende Übertragungsrate, ist somit von geringer Bedeutung.
In herkömmlichen P2P-Netzen werden Peers dynamisch Mitglied im P2P-Netz, und verlassen dies ebenfalls wieder dynamisch. Ein typisches Beispiel hierfür wäre das Ein- bzw. Abschalten eines PC, der über Modem an das Internet angebunden ist. Durch diese u.U. häufig wechselnden Teilnehmer des P2P-Netzes besteht die Gefahr der Separation des P2P-Netzes in kleinere Einheiten, die nicht miteinander verbunden sind. Die in Abbildung 5.3 präsentierte Architektur hat, im Vergleich zu herkömmlichen P2P-Netzen, eine eher statische Architektur. Es ist zwar auch in diesem Fall mit dem Ausscheiden eines Peers aus dem höherschichtigen P2P-Netz bzw. dem Ausfall eines Connecting-Peers zu rechnen; durch das niederschichtige, domänenbegrenzte P2P-Netz wird in diesem Fall aber ein alternativer Connecting-Peer für die entsprechende administrative Domäne angeboten. Dieser übernimmt die bestehenden Verbindungen seines Vorgängers in der höherschichtigen P2P-Architektur. Deshalb ist im höherschichtigen P2P-Netz keine Dynamik aufgrund von Ausfällen einzelner Connecting-Peers zu erwarten. Somit ist lediglich beim Aufbau des unstrukturierten, höherschichtigen P2P-Netzes darauf zu achten, dass das Netz nicht in Teilnetze separiert wird, sondern zusammenhängt.

Das übergeordnete, durchgängige Overlay-Netz konnektiert alle am Fehlertoleranzdienst beteiligten Knoten in einem strukturierten P2P-Overlay-Netz, das die physikalische Netztopologie berücksichtigt, soweit dies ohne die Einbeziehung von Schichten unterhalb der Internetschicht möglich ist. Dieses Overlay-Netz wird im Folgenden genutzt, um die spezifischen Overlay-Netze zur Wegefindung und zur Contentfindung aufzubauen und bereitzustellen.

5.2.2 Basis-Signalisierung

Durch das übergeordnete P2P-Netz sind jeweils Connecting-Peers der administrativen Domänen konnektiert. Jeder der Connecting-Peers ist Teilnehmer in einem niederschichtigen, domänenbegrenzten P2P-Netz. Die Anzahl der Peers im domänenbegrenzten P2P-Netz stellt die Kapazität C_i des Connecting-Peers dar. In einem regelmäßigen Zeitintervall **t** teilt jeder Connecting-Peer allen anderen Connecting-Peers seine IP-Adresse und seine Kapazität mit. Somit liegen nach Ablauf eines Intervalls **t**, jedem Connecting-Peer die IP-Adressen aller im höherschichtigen P2P-Netz beteiligten Peers sowie deren Kapazitäten vor. Jeder der Connecting-Peers führt zu allen anderen Connecting-Peers im Folgenden einen Traceroute [BBCW01] durch. Nach Abschluss der Traceroutes liegen somit jedem Peer des höherschichtigen P2P-Netzes die Entfernungen zu allen anderen Connecting-Peers vor.
Durch die vorliegenden Daten hat somit jeder Connecting-Peer eine Sicht auf einen virtuellen, vollvermaschten Graphen. Dieser Graph ist Ausgangspunkt für den Aufbau der im Folgenden beschriebenen dedizierten Overlay-Netze.

Eigene IP–Adresse	Cluster Head	Kapazität C_i	Distanz $D_\varnothing$
4	4	1	1

Abbildung 5.4: Datenformat innerhalb des P2P-Netzes (in Byte)

Den Aufwand für die Signalisierung innerhalb des übergeordneten P2P-Netzes schätzen wir folgendermaßen ab: Für die Übermittlung der eigenen Daten an alle Connecting-Peers wird ein Datenpaket mit einer Payload von 10 Byte (siehe Abbildung 5.4) benötigt. Inklusive Overhead, durch notwendige Header, beläuft sich die transferierte Datenmenge auf ca. 114 Byte pro Vorgang, abhängig von der Anzahl an TCP-Acknowledgements innerhalb des P2P-Netzes. Pro Peer nehmen wir 5 TCP-Verbindungen innerhalb des höherschichtigen P2P-Netzes an.
Wir gehen davon aus, dass ein Peer Daten die er von anderen Peers erhält an alle mit ihm verbunden Peers (außer dem, der die Daten liefert) weitersendet. Bei einer angenommen Anzahl von 10.000 Connecting-Peers ergibt sich somit ein Maximalwert für das Datenvolumen von ca. **4,56 MByte***. Dieses Datenvolumen hat ein Connecting-Peer im ungünstigsten Fall zu bewältigen bzw. an seine Nachbarn zu versenden. Bedingt durch die Struktur des P2P-Netzes und der Unterdrückung von Mehrfachweiterleitung, wird sich aber im Mittel ein geringerer Wert für jeden einzelnen Peer ergeben.
Für den Maximalwert ergibt sich, bei einem gewählten Intervall **t** von ca. einer Stunde, eine im Mittel benötigte Übertragungsrate von ca. 10 KBit/s für einen Connecting-Peer.

*10.000 (Anzahl Connecting-Peers) ∗ 114 (Byte Datenmenge pro Vorgang) ∗ 4 (Nachbarn) ≈ **4,56 MByte**

Diese doch sehr geringe Übertragungsrate stellt für Netzzwischensysteme sicher kein Problem dar. Bemerkenswert ist in diesem Zusammenhang, dass sich die Anzahl der beteiligten Connecting-Peers lediglich linear auf die benötigte Übertragungsrate auswirkt. Somit stellt auch ein u.U. möglicher Anstieg der beteiligten Connecting-Peers um eine Zehnerpotenz kein Hindernis dar. Die erhaltenen Daten belaufen sich pro Connecting-Peer, unter den oben angegebenen Annahmen, dann auf ca. 50 KByte*.

Erhält ein Connecting-Peer eine neue Adresse eines anderen Connecting-Peers, so führt er ein *Traceroute* zu diesem aus. Gegenwärtig ist in Standardbetriebssystemen ein Traceroute folgendermaßen aufgebaut: Die Traceroute-Quelle schickt UDP-Datenpakete mit der Zieladresse des Traceroutes in das Netz. Dabei wird schrittweise die *Time-to-Live (TTL)*, startend bei dem Wert 1, erhöht. Für jeden TTL-Wert werden drei UDP-Pakete versandt. Auf dem Weg durch das Netz dekrementiert jeder Router das TTL-Feld um eins. Ist die TTL bei dem Wert 0 angelangt, und gegenwärtig auf einem Router im Netz der nicht der Zieladresse des UDP-Pakets entspricht, sendet der Router ein ICMP-Datenpaket an die Traceroute-Quelle mit der Nachricht zurück: *Time-to-Live-Exceeded*.
Kommt das UDP-Paket an der Zieladresse an, sendet der Zielrechner ebenfalls ein ICMP-Datenpaket an die Quelle zurück mit dem Wert: *Destination-unreachable*. Dies resultiert daher, da das UDP-Paket absichtlich einen Port der Zieladresse adressiert, der nicht vorhanden ist.
Dieser Mechanismus ist in das Framework des Fehlertoleranzdienstes integriert. Ein Durchschnittswert für Entfernungen von Rechnern im Internet liegt bei etwa 15 Hops. Die vom Fehlertoleranzdienst generierten UDP-Datenpakete zur Traceroute-Messung haben eine Länge von 68 Byte†. Die von den Routern im Netz zurückgeschickten ICMP-Antworten auf die UDP-Pakete variieren in der Länge‡ und sind abhängig von der jeweiligen Implementierung in den Core-Routern.

Somit berechnet sich das Datenaufkommen für einen Connecting-Peer, der ein Traceroute ausführt, folgendermaßen:

$$H_{(Anzahl-Hops)} * I_{(Identische-TTL)} * (LU_{(Avg-Länge-UDP)} + LI_{(Avg-Länge-ICMP)})$$

Mit den angegebenen Werten ergibt sich somit ein durchschnittliches Datenaufkommen pro Traceroute von ca. 7,4 KByte. Ein Connecting-Peer führt Traceroutes zu allen anderen Connecting-Peers aus. Damit ergibt sich für einen Connecting-Peer in etwa ein Datenaufkommen von ca. **75 MByte****, um diese auszuführen.
Zusätzlich müssen noch Traceroutes der anderen Peers beantwortet werden, was noch einmal mit ca. 5 MByte†† an zu übertragenden Daten zu Buche schlägt.

Insgesamt gehen wir daher von einem Datenvolumen von ca. 80 MByte aus, das von jedem einzelnen Connecting-Peer bewältigt werden muss, um alle Traceroutes zu bedienen. Führt

*10.000 (Anzahl Connecting-Peers) ∗ 5 Byte (IP-Adresse in IPv4 + Kapazität) ≈ 50 KByte

†Dies ist dem gegenwärtigen Standard in Betriebssystemen nachempfunden.

‡In Messungen zeigte sich ein Mittelwert von ca. 100 Byte Paketlänge. Dies liegt daran, dass (implementierungsabhängig) ein Teil des gesendeten UDP-Pakets an das ICMP-Paket angehängt, und somit zurückgeschickt wird.

**10.000 (Anzahl Connecting-Peers) ∗ 7,5 KByte (Größe eines Traceroute) ≈ 75 MByte

††10.000 (Anzahl Connecting-Peers) ∗ 3 (Anzahl ICMP) ∗ 168 Byte (Größe UDP/ICMP) ≈ 5 MByte

man diese Traceroutes ebenfalls verteilt über einen Zeitraum von ca. einer Stunde aus, dann ergibt sich im Mittel eine benötigte Übertragungsrate von ca. 178 KBit/s.

Addiert man die oben angegebene benötigte Übertragungsrate zur Übermittlung der IP-Adressen, so summiert sich die benötigte Übertragungsrate für die Signalisierung im übergeordneten P2P-Overlay-Netz auf ca. 190 KBit/s. Die Größenordung dieser Übertragungsrate ist für Netzzwischensysteme, die in der Regel mindestens über Fast-Ethernet oder schneller, an das Internet angeschlossen sind, sicher kein Problem.

5.3 Dedizierte Overlay-Netze zur Wegefindung

In Abschnitt 5.2.1 ist beschrieben, wie Connecting-Peers Information über alle am Fehlertoleranzdienst beteiligten Connecting-Peers austauschen. Jedem Connecting-Peer liegt nach einem Zeitintervall die Information über alle anderen beteiligten Connecting-Peers vor. Außerdem sind die Distanzen zu diesen, und deren Kapazitäten bekannt. Diese Information wird innerhalb des domänenbegrenzten, niederschichtigen P2P-Netzes verteilt. Somit ist jeder am Fehlertoleranzdienst beteiligte Knoten über alle höherschichtigen Connecting-Peers informiert. Dies ist wichtig für mögliche Ausfälle von Connecting-Peers. Ein anderer Knoten in der Domäne eines Connecting-Peers kann dann schnell dessen Funktion bei einem Ausfall übernehmen. Außerdem ist dieses in den Domänen vorhandene Wissen die Grundlage für den Aufbau der Overlay-Netze zur Wegefindung bzw. zur Contentfindung.
Im Folgenden beschreiben wir den Aufbau der Overlay-Netze zur Wegefindung und deren Funktionsweise.

5.3.1 Aufbauprozess

Das in Abbildung 5.4 beschriebene Format für den Austausch von Informationen im höherschichtigen P2P-Netz zwischen den Connecting-Peers, wird ebenfalls zum Aufbau der dedizierten Overlay-Netze zur Wegefindung benützt.
Wir gehen von einem allmählichen Netzaufbau bzw. von einer allmählichen Verbreitung des Fehlertoleranzdienstes aus. Wir nehmen an, dass in einer administrativen Domäne einige Zugangsknoten für den Fehlertoleranzdienst bereitgestellt werden. Diese konnektieren sich im niederschichtigen, domänenbegrenzten P2P-Netz und legen einen Connecting-Peer fest. Dieser Connecting-Peer verbindet sich dann im höherschichtigen P2P-Netz und tauscht, wie in Abschnitt 5.2.2 beschrieben, Informationen mit anderen Connecting-Peers aus bzw. ermittelt die Distanzen mittels Traceroutes.

Der erste Connecting-Peer, der initialisiert wird, kann sich natürlich noch nicht mit anderen Connecting-Peers verbinden, da diese noch nicht existieren. Bei einer erfolgreichen Akzeptanz des Dienstes gehen wir davon aus, dass weitere Connecting-Peers in anderen Domänen folgen werden. Diese sind dann über das höherschichtige P2P-Netz verbunden.

Um Skalierbarkeit bei der Wegesuche bzw. -findung zu gewährleisten, führen wir, unabhängig zum höherschichtigen P2P-Netz, weitere Overlay-Netze ein. Diese fassen Connecting-Peers bzw. administrative Domänen zu *Clustern* zusammen.

Parallel zum Aufbau des höherschichtigen P2P-Netzes erfolgt die Zusammenfassung von Connecting-Peers in Cluster. Grundprinzip der Clusterbildung ist Folgendes:

1. Aufnahme eines neuen Connecting-Peers in einen bestehenden Cluster.
2. Wächst eine Clustergröße auf die Anzahl von 11 beteiligten Connecting-Peers* an, dann wird der Cluster geteilt: In einen neuen Cluster mit fünf und einen weiteren mit sechs Connecting-Peers.

Im Wesentlichen gibt es bei der Clusterbildung drei Punkte zu beachten:

- Wie messe ich die Güte eines Clusters?
- In welchen (bestehenden) Cluster soll ein neuer Connecting-Peer aufgenommen werden?
- Wie teilt man einen bestehenden Cluster in zwei kleinere Cluster auf?

Auf diese Fragestellungen gehen wir in den folgenden Abschnitten ein.

5.3.2 Güte eines Clusters

Generelles Ziel eines Clusters von Connecting-Peers ist es, angelehnt an das *Resilient Overlay Networks*-Projekt (RON) [ABKM01], alternative Pfade für Datenverbindungen im Internet zur Verfügung zu stellen. Innerhalb des Clusters wird ein Knoten ausgewählt, der als Teil eines Overlay-Netzes fungiert, welches einen alternativen Pfad zur Verfügung stellt.
Damit dies allgemein für alle Verbindungen gut funktioniert, ist es wichtig, dass die zur Verfügung stehenden Knoten im Internet topologisch verteilt sind (siehe Anhang A). Unter diesem Aspekt verstehen wir den Begriff der Güte eines Cluster. Als Parameter führen wir die durchschnittliche Distanz $D_\emptyset$ zwischen den Connecting-Peers eines Clusters ein.

Durch die Basis-Signalisierung (siehe Abschnitt 5.2.2), sind jedem Connecting-Peer die Distanzen zu den Connecting-Peers seines Clusters bekannt.
Diese Informationen werden zwischen allen Connecting-Peers innerhalb eines Clusters ausgetauscht. Das Datenaufkommen beträgt in diesem Fall pro Connecting-Peer in etwa 50 Byte. Am Datenaustausch in einem Clusters sind höchstens 10 Connecting-Peers beteiligt; Skalierbarkeit ist somit gegeben.
Im Anschluss errechnet jeder Connecting-Peer die durchschnittliche Distanz $D_\emptyset$ in seinem Cluster. Diese berechnet sich folgendermaßen:

$$D_\emptyset = \frac{2}{n(n-1)} \sum_{j=1}^{\frac{n(n-1)}{2}} d_j \qquad n \quad \text{Anzahl der Knoten im Cluster}$$

Die gemittelte Distanz (in ms) zweier Connecting-Peers wird durch d_j angegeben. Damit ist gemeint, die Mittelung der Distanzen in Hin- und Rückrichtung. Wir gehen somit von einem ungerichteten, virtuellen und vollvermaschten Graphen in einem Cluster aus.

*Anzahl angelehnt an empirische Messungen in RON.

Die Güte eines Clusters kann man nun folgendermaßen deuten: Liegen die beteiligten Connecting-Peers topologisch eng zusammen, dann ist $D_{\emptyset}$ klein. Sind die beteiligten Peers im Netz verteilt, ist $D_{\emptyset}$ größer.

Ist eine Clusterbildung abgeschlossen, werden die Informationen über das höherschichtige P2P-Netz, wie in Abbildung 5.4 beschrieben, an alle Connecting-Peers (auch ausserhalb des Clusters) verteilt. Ein Knoten innerhalb des Clusters wird zum *Cluster-Head* bestimmt. Dieser Knoten hat keine weitere Funktionalität. Seine IP-Adresse dient lediglich zur eindeutigen Identifizierung des Clusters (siehe Abbildung 5.4). Jeder Connecting-Peer hat die durchschnittliche Distanz $D_{\emptyset}$ seines Clusters berechnet. Des Weiteren steht jedem Connecting-Peer über das niederschichtige P2P-Netz die Information zur Verfügung, wie viele Knoten in seiner Domäne zur Verfügung stehen. Dies entspricht der Kapazität C_i des Connecting-Peers.
Diese Informationen (Cluster-Head, Kapazität C_i, Distanz $D_{\emptyset}$) werden über die kontinuierlich ablaufende Signalisierung im höherschichtigen P2P-Netz an alle Connecting-Peers verteilt, wie in Abschnitt 5.2.2 beschrieben.

5.3.3 Aufnahme in einen Cluster

Eine neue Domäne, die sich dem Fehlertoleranzdienst anschließen will, baut zunächst ein niederschichtiges P2P-Netz auf und bestimmt einen Connecting-Peer. Dieser wird Teilnehmer im höherschichtigen P2P-Netz und sammelt Informationen durch die Basis-Signalisierung. Nach einer Weile stehen dem Connecting-Peer die beschriebenen Informationen zur Verfügung. Das sind insbesondere die Daten über die einzelnen Cluster und deren Teilnehmer. Dem neuen Connecting-Peer sind auch die Distanzen zu allen anderen Connecting-Peers bekannt.
Für alle bestehenden Cluster berechnet der neue Connecting-Peer dann die neue durchschnittliche Distanz $D_{\emptyset Neu}$, für den Fall, dass er Mitglied im jeweiligen Cluster würde:

$$D_{\emptyset Neu} = ((D_{\emptyset} * \frac{n(n-1)}{2}) + \sum_{i=\frac{n(n-1)}{2}}^{\frac{n(n+1)}{2}} d_{iNeu}) * \frac{2}{n(n+1)}$$

Die Distanzen des neuen Connecting-Peers zu den beteiligten Connecting-Peers eines Clusters werden dabei durch d_{iNeu} angegeben. Eine Mittelung der Distanzen in Hin- und Rückrichtung findet nicht statt, da sonst eine zusätzliche Signalisierung notwendig wäre.
Der berechnete Wert $D_{\emptyset Neu}$ ist somit nur ein Näherungswert für die tatsächliche mittlere Distanz eines neuen Clusters. Der Vorteil liegt darin, dass zur Berechung von $D_{\emptyset Neu}$, abgesehen vom ohnehin im höherschichtigen P2P-Netz ablaufenden Datentransport, kein weiterer Datenaustausch notwendig ist.
Die Näherungswerte $D_{\emptyset Neu}$ zu den mittleren Distanzen für alle Cluster können somit lokal – und daher sehr schnell – im neuen Connecting-Peer berechnet werden, ohne weitere Interaktion mit dem Netz.

Für jeden Cluster berechnet der neue Peer die Differenz $\text{Diff}_{\emptyset}$ des Näherungswertes der neuen durchschnittlichen Distanz zur bisherigen durchschnittlichen Distanz innerhalb eines Cluster:

$$\text{Diff}_{\emptyset} = \text{D}_{\emptyset Neu} - \text{D}_{\emptyset}$$

Des Weiteren berechnet der neue Connecting-Peer noch die durchschnittliche Kapazität jedes Clusters. Auch dies ist ohne weitere Signalisierung möglich.
Aus den Clustern deren durchschnittliche Kapazität in etwa dem des neuen Connecting-Peers entspricht, wählt er den Cluster mit dem größten Wert für die Differenz $\text{Diff}_{\emptyset}$ aus. Diesen Cluster kontaktiert der neue Connecting-Peer und bittet um seine Aufnahme.

5.3.4 Datenaustausch innerhalb eines Clusters

Jedem Connecting-Peer liegt, durch die in Abschnitt 5.2.2 beschriebene traceroute-basierte Signalisierung, die Information über die Pfade zu allen anderen Connecting-Peers vor. Das lokal zu speichernde Datenaufkommen beträgt hierfür, bei angenommener Teilnahme von 10.000 Connecting-Peers im höherschichtigen P2P-Netz, in etwa **800 KByte**[*] pro Connecting-Peer. Innerhalb eines Clusters werden diese Informationen zwischen den Connecting-Peers ausgetauscht. Diese verbreiten die Information an niederschichtige Peers in ihrer Domäne. Somit verfügt jeder Knoten einer Domäne über folgende Informationen:

- Pfad vom Connecting-Peer in seiner Domäne zu den *anderen Connecting-Peers* im zugehörigen Cluster.
- Pfad von den *anderen Connecting-Peers seines Clusters*, zu allen Connecting-Peers des höherschichtigen P2P-Netzes.

Das gesamte, für den Datenaustausch innerhalb des Clusters, anfallende Datenaufkommen beträgt im Mittel ca. 6 MByte[†] pro Connecting-Peer. Diese Daten müssen ebenfalls in dem in Abschnitt 5.2.2 eingeführten Zeitintervall **t** übertragen werden. Die dafür zusätzlich anfallende Signalisierung beläuft sich somit auf ca. 13 KBit/s pro Connecting-Peer.
Des Weiteren verteilt der Connecting-Peer die gesammelten Informationen innerhalb seiner Domäne an die entsprechenden niederschichtigen Knoten weiter. Hierfür kommt noch einmal ein Wert in ähnlicher Größenordnung dazu.
Für jeden Connecting-Peer eines Clusters fällt somit durch den Datenaustausch innerhalb eines Clusters insgesamt eine Signalisierung mit einer Übertragungsrate von ca. 30 KBit/s an.

5.3.5 Aufspaltung eines Clusters

Durch die in den Abschnitten 5.3.2 und 5.3.3 beschriebenen Mechanismen nimmt die Zahl der Teilnehmer in einem Cluster stetig zu. Die in Abschnitt 5.3.1 formulierte Grenze liegt

[*]Pro Traceroute ca. 16 IP-Adressen bzw. 80 Byte an Daten. 10.000*80 Byte = 800 KByte.
[†]7,5 (Teilnehmer) * 800 KByte = 6 MByte

bei 10 teilnehmenden Connecting-Peers pro Cluster. Wird ein weiterer Connecting-Peer aufgenommen, dann wird der bestehende Cluster geteilt. Es entstehen zwei neue Cluster mit 5 bzw. 6 Teilnehmern.
In diesem Abschnitt beschreiben wir den Vorgang der Teilung. Innerhalb eines zu teilenden Clusters liegt ein vollvermaschter, virtueller Graph mit 11 Knoten vor. Durch die Vollvermaschung ergibt sich eine Kantenzahl von 50, mit entsprechender durchschnittlicher Distanz $D_{\emptyset}$. Ein weiterer zur Verfügung stehender Parameter ist die durchschnittliche Kapazität des Clusters. Ziel ist es, diesen Cluster in zwei neue Cluster aufzuteilen, unter Berücksichtigung der neuen durchschnittlichen Distanzen bzw. Kapazitäten.
Eine weitere Randbedingung ist die Aufteilung in zwei annähernd gleich große Cluster mit 5 bzw. 6 Teilnehmern. Ein großes Ungleichgewicht in der Zahl beteiligter Knoten würde zu Einschränkungen für den im nächsten Abschnitt beschrieben Algorithmus der Wegesuche führen.

Somit ergeben sich prinzipiell $\binom{11}{5} = 462$ Möglichkeiten, den Cluster aufzuteilen. Bedingt durch diese relativ kleine Menge an möglichen Lösungen, können für jeden Fall folgende Parameter der zwei neuen Cluster berechnet werden:

- Durchschnittliche Distanzen: $D_{\emptyset 1}$, $D_{\emptyset 2}$
- Varianz der Kapazitäten: $c_{\emptyset 1}$, $c_{\emptyset 2}$

Aus der Lösungsmenge wird diejenige Teilung des Clusters gewählt, welche die höchsten durchschnittlichen Distanzen aufweist, bei gleichzeitig geringer Varianz der jeweiligen Kapazitäten.

5.3.6 Zeitliche Betrachtung des Aufbaus

Wie in Abschnitt 5.2.2 angeführt, werden innerhalb des höherschichtigen P2P-Netzes, bei ubiquitärer Verbreitung des Fehlertoleranzdienstes, in einem Zeitintervall $\mathbf{t}$ von ca. einer Stunde, die in Abbildung 5.4 dargestellten Informationen zwischen allen Connecting-Peers ausgetauscht. Dies geschieht bei relativ moderaten Datenraten für die einzelnen Connecting-Peers.
Zu Beginn des Netzaufbaus ist das benötigte Zeitintervall $\mathbf{t}$ geringer. Bei einer angenommenen allmählichen Verbreitung des Dienstes, sind anfangs wenige Connecting-Peers im Netz vorhanden. Somit ist die benötigte Zeit für die Verteilung der Daten kleiner.

Hat ein noch nicht in einen Cluster integrierter, neuer Connecting-Peer alle Informationen über das höherschichtige P2P-Netz empfangen, startet er, wie in Abschnitt 5.3.3 angegeben, lokale Berechnungen. Im Anschluss steht dem Knoten eine priorisierte Liste an möglichen Clustern zur Verfügung.
Diese Cluster werden kontaktiert und der gegenwärtige Stand ermittelt. Das ist notwendig, da sich die Cluster bereits verändert haben können. Hat der angefragte Cluster keine (oder nur geringe) Abweichungen zu den vorausgegangenen Berechnungen, dann wird der anfragende Connecting-Peer Mitglied des Clusters.

Ein Cluster kann maximal bis zu sechs neue Teilnehmer aufnehmen; dann wird er geteilt. Um die Daten in allen Connecting-Peers auf dem aktuellen Stand zu halten, muss nach der

erfolgten Teilung des Clusters die Information erst im höherschichtigen P2P-Netz verteilt werden. Somit können maximal 6 Connecting-Peers pro Cluster im Zeitintervall t aufgenommen werden.
Zu Beginn des Netzaufbaus stehen nicht viele Connecting-Peers zur Verfügung. Aus diesem Grund können insgesamt nur wenig neue Connecting-Peers pro Zeitintervall t aufgenommen werden. Im Gegenzug bedingt eine geringe Zahl an vorhandenen Connecting-Peers jedoch ein kleines Zeitintervall t.
Bei annähernd ubiquitärer Verbreitung des Fehlertoleranzdienstes beträgt das erwartete Zeitintervall t ca. eine Stunde. In diesem Fall existiert bereits eine große Anzahl an Clustern, in die viele neue Connecting-Peers parallel aufgenommen werden können.
Bei einer angenommenen durchschnittlichen Rate von 20 neuen Connecting-Peers pro Stunde, die in einen der Cluster aufgenommen werden wollen, würde somit ein Netzaufbau, bei ubiquitärer Verbreitung des Fehlertoleranzdienstes, in etwa drei Wochen in Anspruch nehmen. Diese, auf den ersten Blick, sehr lange Zeit relativiert sich, wenn man den notwendigen Aufwand der Bereitstellung berücksichtigt: Es müssen in jeder Domäne an entsprechenden Knoten Änderungen vorgenommen werden. Allein der Zeitaufwand, bis z.B. nur eine Domäne entsprechend konfiguriert ist (Hardware- und Software-Tests), nimmt vermutlich einige Tage bis Wochen in Anspruch.
Somit ist ein zeitlich realistisches Szenario – von der ersten Inbetriebnahme bis zu einer vollständigen Verbreitung im Internet – eher in Monaten oder länger zu sehen. Diesen Anforderungen genügt der beschriebene Prozess des Netzaufbaus bei weitem. Der Vorteil der Architektur liegt darin, dass selbst bei nur wenigen installierten Knoten im Netz, der Fehlertoleranzdienst – zwar mit eingeschränkter Funktionalität – sofort in Betrieb genommen werden kann.

5.3.7 Auflösen von Clustern

Durch den beschriebenen Mechanismus sollen Cluster bis zu einer gewissen Größe anwachsen, werden dann geteilt, und das Wachsen beginnt erneut. Ziel ist es, dass jeder Cluster eine hohe durchschnittliche Distanz $D_{\emptyset}$ und eine niedere Varianz der Kapazität $c_{\emptyset}$ hat.
Im Wesentlichen kann es zu zwei Fehlentwicklungen kommen:

- Obwohl ein Cluster nicht wächst, kann es zu Vergrößerungen von $c_{\emptyset}$ kommen.
- Niedrige und konstant bleibende $D_{\emptyset}$ eines Clusters über einen langen Zeitraum.

Ist ein Cluster gebildet, kann es sein, dass innerhalb einer Domäne zusätzliche Knoten installiert bzw. Knoten abgeschaltet werden; Dies verändert die Kapazität des Connecting-Peers und somit auch die Varianz $c_{\emptyset}$ des entsprechenden Clusters. Überspringt die Varianz einen gewissen Höchstwert, dann liegt u.U. ein erhebliches Ungleichgewicht im Cluster vor.
Hat ein Cluster eine sehr geringe Distanz $D_{\emptyset}$, im Vergleich zu anderen Clustern, dann ist es unwahrscheinlich, dass ein neuer Connecting-Peer diesem Cluster beitritt. Folglich bleibt $D_{\emptyset}$ u.U. lange Zeit auf diesem geringen Wert. Die durchschnittliche Distanz ist ein Indikator dafür, wie eng die im Cluster beteiligten Connecting-Peers topologisch zusammenliegen. Ziel ist eine hohe $D_{\emptyset}$, um bessere alternative Routen anbieten zu können.

In beiden Fällen schlagen wir deshalb vor, derartige Cluster nach einer gewissen Wartezeit aufzulösen. Die einzelnen Connecting-Peers agieren dann als *neue* Connecting-Peers und suchen sich jeweils, wie in Abschnitt 5.3.3 beschrieben, einen neuen Cluster.

5.3.8 Dynamische RON durch kooperierende Cluster

In diesem Abschnitt erläutern wir exemplarisch die Suche und Bereitstellung eines alternativen Weges durch den Fehlertoleranzdienst. Die in Abschnitt 5.3.1 beschriebenen Cluster sind dafür die Grundlage. Durch den eingeführten Mechanismus der Clusterbildung ist zu erwarten, dass ein Connecting-Peer im Mittel mit ca. 7-8 anderen Connecting-Peers in einem Cluster zusammengefasst ist.

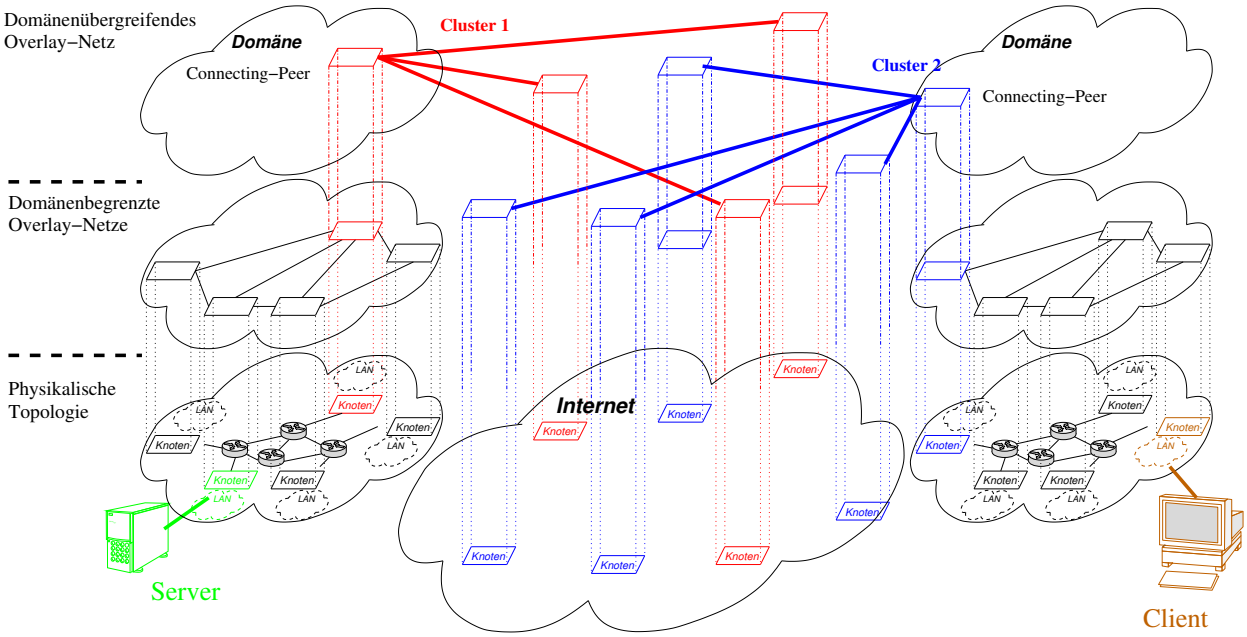

Abbildung 5.5: Wegesuche in dynamischem RON

Wie in Abbildung 5.5 dargestellt, gehen wir von einer gewünschten Kommunikationsverbindung zwischen einem Client und einem Server aus. Beide Kommunikationspartner befinden sich in unterschiedlichen Domänen. Die Domänen sind über das Internet verbunden.
In jeder Domäne wurde ein niederschichtiges P2P-Netz zwischen den am Fehlertoleranzdienst beteiligten Knoten aufgebaut und jeweils ein Connecting-Peer gewählt. Diese sind in das höherschichtige P2P-Netz eingebunden. Die beschriebene Clusterbildung hat ebenfalls stattgefunden. Die zwei Connecting-Peers sind Teilnehmer in unterschiedlichen Clustern: Cluster 1 und 2.
Der Client initiiert einen Verbindungswunsch zum Server und beginnt z.B. einen TCP-Handshake. Der verantwortliche Knoten des Fehlertoleranzdienstes, der den Client mit der Domäne verbindet, registriert dies.

Durch zusätzlich ausgetauschte Datenpakete identifizieren sich der clientseitige Knoten und der serverseitige Knoten der den Server an seine Domäne konnektiert. Des Weiteren werden Informationen ausgetauscht über die jeweiligen Connecting-Peers, und somit sind die

beteiligten Cluster identifiziert. Wie in Abschnitt 5.3.3 beschrieben, liegen sowohl dem serverseitigen als auch dem clientseitigen Knoten Informationen über folgende Pfade, ohne weitere Signalisierung, vor:

- Pfade vom serverseitigen bzw. clientseitigen Connecting-Peer zu deren Teilnehmern in den entsprechenden Clustern.
- Pfade von den Teilnehmern in den Clustern zu serverseitigem *und* clientseitigem Knoten.

Aufgrund des Aufbaus der Domänen nehmen wir an, dass die Pfade vom Connecting-Peer einer Domäne zu den jeweiligen Teilnehmern seines Clusters (Connecting-Peers), den Pfaden zwischen den niederschichtigen Knoten einer Domäne zu den Knoten der im Cluster beteiligten anderen Domänen stark ähneln.
Basierend auf dieser Annahme liegen dem serverseitigen- bzw. clientseitigen Knoten alle notwendigen Informationen vor, um mögliche alternative Pfade zu berechnen:

- Als direkten Pfad zwischen client- und serverseitigem Knoten nehmen wir den bekannten Pfad zwischen den Connecting-Peers der jeweiligen Domänen her.
- Die zu untersuchenden, möglichen alternativen Pfade bestehen aus Overlay-Netzen mit jeweils drei Hops: Connecting-Peer der Domäne des Servers, Connecting-Peer der Domäne des Clients; und ein weiterer Connecting-Peer aus einem der beiden Cluster, der als Relay dient.

Auf den vorliegenden Informationen aufbauend, können der serverseitige und der clientseitige Knoten jeweils die zusammengesetzten alternativen Routen mit der direkten Route vergleichen. Somit entsteht eine priorisierte Liste der alternativen Pfade.
Einer der alternativen Pfade wird gewählt. Der entsprechende Connecting-Peer, dessen Domäne als Relay dienen soll, wird kontaktiert mit der Bitte, einen Knoten in seiner Domäne als Relay zur Verfügung zu stellen. Dieser Knoten wird dann als Relay konfiguriert. Im Anschluss daran steht ein alternativer Datenpfad für die gewünschte Verbindung zwischen Client und Server zur Verfügung.

Der in diesem Kapitel beschriebene Mechanismus der Clusterbildung erhebt nicht den Anspruch, die optimalen Cluster, hinsichtlich der Verteilung im Netz, zu finden. Ziel ist die skalierbare Unterteilung des in Abschnitt 5.2.1 beschriebenen, übergeordneten P2P-Netzes, unter Berücksichtigung der einzelnen Kapazitäten. Dabei beziehen wir auch die Verteilung, durch die Einführung der Cluster-Güte, mit ein.
Die in RON [ABKM01] empirisch untermauerte Aussage, dass selbst eine zufällige Verteilung entsprechender Overlay-Relays im Internet bereits genügt, um alternative Pfade zur Verfügung zu stellen, wurde von uns in Messungen nachvollzogen (siehe Anhang A). Dies ist Voraussetzung für diesen Abschnitt.

Die Wegefindung wurde in diesem Abschnitt für Unicast-Datenverkehr erläutert. Grundlage ist eine reguläre Client-Server-Kommunikation. Der Mechanismus kann auch zur Findung von alternativen Wegen bzw. Bäumen benutzt werden, wie in Diverse-Protected-Multicast [BT03] beschrieben. Lediglich die Detektion des originalen Datenweges benötigt einen an die jeweilige Multicast-Technologie angepassten Traceroute.

5.3.9 Skalierbarkeit

Bezugnehmend auf die in Abbildung 5.3 dargestellte Architektur nehmen wir an, dass innerhalb einer administrativen Domäne maximal 256 Knoten am Fehlertoleranzdienst teilnehmen. Diese Knoten wählen einen Connecting-Peer. Zusätzlich nehmen wir an, dass jeder Knoten ein LAN mit dem Internet verbindet, aus dem im Mittel 256 neue Verbindungen in das Internet pro Minute aufgebaut werden. Dies entspricht dem Fall, dass 256 Endsysteme im LAN konnektiert sind, die im Mittel eine neue Verbindung pro Minute über das Internet aufbauen. Somit ergeben sich innerhalb der administrativen Domäne im Mittel 65536 neue Verbindungen über das Internet pro Minute. Pro Sekunde gehen wir daher von ca. 1100 neuen Verbindungen pro Domäne über das Internet aus.

Die in den Abschnitten 5.2.2 und 5.3.4 dargestellte Signalisierung ist unabhängig von der Zahl der durch den Fehlertoleranzdienst behandelten Verbindungen.
Der Aufwand hingegen, für die in Abschnitt 5.3.8 dargestellte Signalisierung, ist abhängig von der Anzahl der neuen Verbindungen pro Zeiteinheit: Die Suche nach und der Aufbau von alternativen Pfaden erfolgt für jede neue Verbindung.
In diesem Abschnitt werden die beteiligten niederschichtigen Knoten und der Connecting-Peer, dessen Domäne als Relay gewählt wird, näher betrachtet.

In den niederschichtigen Knoten nehmen wir in etwa 4 neue Verbindungen pro Sekunde an, die von Endsystemen im konnektierten LAN mit dem Internet über einen Zugangsknoten initiiert werden. Der lokal ausgewählte, entfernte Connecting-Peer, dessen Domäne als Relay fungieren soll, wird durch den Zugangsknoten direkt kontaktiert. Dieser liefert die Adresse des Relay, eines entsprechenden Knoten seiner Domäne zurück. Im Anschluss wird das Relay kontaktiert und konfiguriert. Pro Verbindung werden insgesamt ca. 30 Byte* an Nutzdaten signalisiert. Dies ergibt, inklusive Overhead, in etwa 210 Byte† an Datenaufkommen pro Verbindung. Hiermit ergibt sich für die niederschichtigen Knoten ein Signalisierungsaufwand von kleiner 10 KBit/s‡.

Innerhalb eines Clusters ergeben sich im Mittel ca. 8000 angenommene neue Verbindungen pro Sekunde. Die Connecting-Peers eines Clusters (bzw. Knoten in deren Domäne) sollen für die Verbindungen als Relay dienen. Somit sind ca. 1000 Anfragen pro Sekunde zu erwarten, die von niederschichtigen Knoten an Connecting-Peers nach einer Relay-Adresse in deren Domäne gestellt werden. Ausgetauscht werden jeweils 4 Byte an Nutzdaten. Die notwendige Signalisierung seitens der Connecting-Peers beträgt dafür in etwa 640 KBit/s**, inklusive Overhead.

In Relation dazu betrachten wir das Volumen der Nutzdaten, bzw. der zu schützenden Flows, die einen niederschichtigen Knoten passieren. In unserer Annahme gehen wir von vier neuen Verbindungen pro Sekunde innerhalb des konnektierten LAN aus. Nehmen wir weiter an, dass jede dieser Verbindungen im Mittel 60 Sekunden besteht und eine durchschnittliche Übertragungskapazität von 300 KBit pro Sekunde benötigt. So ergäbe das dann

*Adresse des Relays; Daten des zu schützenden Flows.
†Zweimal UDP-basierter Austausch von Nutzdaten.
‡Datenvolumen pro Sekunde: 210 Byte $*$ 4 $\approx$ 6,7 KBit
**80 Byte $*$ 1000 $\approx$ 640 KBit

eine gesamte, durchschnittliche Übertragungsrate von etwa 70 MBit/s für den Zugangsknoten.
Die im vorrangehenden, abgeschätzten notwendigen Übertragungsraten zur Signalisierung des Fehlertoleranzdienstes sind in etwa um zwei Größenordnungen kleiner:

Die notwendige Übertragungsrate zur Signalisierung der Wegefindung ist somit, im Vergleich zum Nutzdatenaufkommen, vernachlässigbar.

Deshalb betrachten wir im Folgenden nur noch die benötigten Übertragungsraten bei der Übertragung bzw. dem Ersatzschalten von Nutzdaten. Ist der alternative Pfad durch das Netz aufgebaut, bzw. das in Abschnitt 5.5 näher beschriebene Overlay-Netz für den Datentransport konfiguriert, können prinzipiell folgende Ersatzschaltmechanismen angewandt werden:

- 1+1 Ersatzschalten
- 1:1 Ersatzschalten

Das 1+1 Ersatzschalten ermöglicht einen unterbrechungsfreien Betrieb. Beim 1:1 Ersatzschalten muss ein Ausfall erst detektiert werden. Nach der dafür benötigten Zeit wird der, bereits vorkonfigurierte, alternative Pfad benutzt. Erst dann steht wieder Konnektivität für die Applikationen zur Verfügung.
Der Nachteil des 1+1 Ersatzschaltens liegt in dem u.U. verschwenderischen Umgang mit Ressourcen. Selbst für den Fall, dass während der gesamten Laufzeit einer Verbindung kein Ausfall auftritt, werden Daten doppelt im Netz übertragen. Im Vergleich dazu: Das 1:1 Ersatzschalten würde hier keinerlei Daten doppelt übertragen.

Das vorgeschlagene Prinzip des Fehlertoleranzdienstes lehnt sich an das Prinzip der File-Sharing P2P-Netze (siehe Abschnitt 2.2.4) an. Grundprinzip in File-Sharing P2P-Netzen ist, dass jeder Teilnehmer anderen Teilnehmern seine lokalen Daten zur Verfügung stellt. Des Weiteren hilft jeder Teilnehmer den anderen Teilnehmern bei der Suche nach bestimmten, im P2P-Netz vorhanden, Daten. Ein Teilnehmer stellt somit seine Ressourcen den anderen Teilnehmern zur Verfügung. Für diese erbrachte Leistung darf der Teilnehmer ebenfalls die Ressourcen der anderen Teilnehmer benutzen. Sucht der betreffende Teilnehmer selbst nach Daten im P2P-Netz, so wird er dabei ebenfalls von den anderen Teilnehmern unterstützt.
Dieses Prinzip der gegenseitigen Unterstützung wenden wir im beschriebenen Fehlertoleranzdienst ebenfalls an. Die in Abbildung 5.5 beteiligten niederschichtigen Knoten, die jeweils Client und Server an das Internet anbinden, benutzen die Ressourcen von anderen Knoten: Zum einen werden die zur Verfügung gestellten Ressourcen genutzt, um einen alternativen Pfad durch das Netz für die Kommunikation zwischen Client und Server zu finden. Zum anderen hinaus wird ein gefundener Pfad durch einen Knoten zur Verfügung gestellt, der beim Datentransport als Relay benutzt wird.

Nehmen wir an, dass jede Kommunikationsverbindung durch den Fehlertoleranzdienst mit einem 1+1 Ersatzschaltverfahren gegen einen Ausfall geschützt ist, bedeutet das Grundprinzip der gegenseitigen Hilfe für einen einzelnen Knoten folgendes:

- Alle vom LAN ausgehenden Verbindungen müssen doppelt übertragen werden.
- Im Relay-Betrieb bedeutet jede Verbindung mit der Datenrate r, eine Belastung des Knotens mit 2∗r.

Nehmen wir gleiche Kapazitäten der einzelnen Domänen pro Cluster an, und eine Gleichverteilung der Datenraten aller Verbindungen in einem Cluster, bedeutet das für den einzelnen Knoten in etwa eine Vervierfachung des Datenaufkommens, das zu übertragen ist. Ein mögliches Ungleichgewicht der Kapazitäten innerhalb eines Clusters bzw. eine hohe Varianz der Kapazitäten würde dazu führen, dass einige Knoten im Relay-Betrieb weniger belastet werden, andere Knoten hingegen stärker. Aus diesem Grund werden beim Clusteraufbau die Kapazitäten berücksichtigt.
Zusätzlich wird die Überlastung einzelner administrativer Domänen unterbunden: Alle Anfragen, die Zugangsknoten einer administrativen Domäne als ein Relay benutzen wollen, gehen über den entsprechenden Connecting-Peer. Ist ein Schwellwert überschritten, dann lehnt dieser weitere Anfragen ab. Es muss dann eine andere Domäne als Relay benutzt werden.
Eine Vervierfachung des Datenverkehrs pro Knoten wäre bei idealen Bedingungen gegeben. Dies erscheint jedoch nicht realistisch. In einem realen Szenario sind entsprechende Ungleichgewichte der Kapazitäten zur erwarten. Des Weiteren spielen Eigenschaften des Internetverkehrs, wie Selbstähnlichkeit und Abhängigkeit von der Tageszeit, eine Rolle. Aus diesem Grund ist es sinnvoll, das zu erwartende, maximale Datenaufkommen pro Knoten höher anzusetzen, um kurzfristige Spitzen abfangen zu könnnen.

Während eine Erhöhung der Datenrate, die ein Knoten bewältigen kann, direkt mit höheren Kosten verbunden sein wird, gibt es noch eine weitere Möglichkeit: Da die benötigte Datenrate zum überwiegenden Teil nur vom Transport der Nutzdaten abhängt, und somit die Signalisierung eine untergeordnete Rolle spielt, kann der Einsatz von 1:1 Ersatzschalten die benötigte Datenrate pro Knoten erheblich einschränken. Eine mögliche Verfahrensweise, die dem Prinzip des P2P entspricht, wäre die Folgende: Beansprucht ein Knoten für sich, dass er Verbindungen durch 1+1 Ersatzschaltverfahren schützen will, dann muss er auch als Relay in 1+1 Ersatzschaltverfahren betrieben werden können. Kann er dies jedoch nicht, so wird ihm von den anderen Knoten nur ein 1:1 Ersatzschaltmechanismus angeboten.

Fragestellungen dieser Art sind für den eigentlichen Betrieb des Dienstes von großer Bedeutung. Einhergehend ist damit z.B. auch die Frage des Betreibermodells (Die angenommene große Zahl an beteiligten Knoten würde vermutlich von mehreren kooperierenden Dienstbetreibern zur Verfügung gestellt). Letztlich können derartige Problemstellungen erst im Laufe einer Inbetriebnahme vollständig beantwortet werden. Hierfür ist der Ansatz dieser Arbeit, den Fehlertoleranzdienst in Programmierbaren Netzen anzubieten, von Vorteil. Somit bleibt eine Flexibilität vorhanden, mit der man den verteilten Fehlertoleranzdienst relativ schnell an neue Gegebenheiten bzw. Betreibermodelle, anpassen kann.

5.4 Dedizierte Overlay-Netze zur Bereitstellung alternativen Contents

Die wesentliche Funktionalität des, in diesem Kapitel beschriebenen Fehlertoleranzdienstes, liegt auf dem in Abschnitt 5.3 dargelegten Ersatzschalten durch Overlay-Netze.
Die in den Abschnitten 4.1.3 und 4.1.4 beschriebenen Fehlermodelle des Unicast- bzw. Gruppenkommunikation können damit gegen Ausfälle im Netz geschützt werden.

In diesem Abschnitt gehen wir auf Ausfälle der Endsysteme ein. Wir beschränken uns dabei auf die Betrachtung von Server- bzw. Contentausfällen im Rahmen einer Client-Server-Kommunikation. Wesentlich ist dabei der in Abschnitt 4.1.5 eingeführte Applikationsfehler. Wir gehen davon aus, dass ein Endsystem einen bestimmten Content aus dem Netz anfragt. Dieser ist definiert durch die IP-Adresse des Servers, und einen entsprechenden, eindeutigen Namen. Betrachtet man die Applikationsschicht des anfragenden Client, dann ist es für einen fehlerfreien Betrieb der Applikation Voraussetzung, dass diese genau den eindeutig identifizierten Content aus dem Netz erhält. Ob dieser Content tatsächlich vom angegebenen Server kommt, ist dabei für die Funktionalität der Applikation unwichtig. Dieses Prinzip nutzen die in Abschnitt 3.3.2 beschriebenen Content-Distribution-Netze (CDN), bzw. Cache-Systeme. Ziele dieser Architekturen sind im Wesentlichen die Verbesserung der Skalierbarkeit von Servern und die Minimierung der Übertragung von Daten durch das Netz. In diesem Abschnitt nutzen wir das Grundprinzip, dass Content auch an alternativen Stellen im Netz zur Verfügung stehen kann, zur Minimierung von Applikationsfehlern. Wir präsentieren eine Architektur kollaborierender Caches, die es erlaubt, bei einem Ausfall eines originalen Servers, einen angefragten Content von alternativen Stellen im Netz zu beziehen.

Ausfälle von Servern können durch die in Abschnitt 3.5.1 beschriebenen, protokollbasierten Fehlertoleranzmechanismen in Endsystemen, wie FT-TCP oder ST-TCP, maskiert werden. Dabei werden Server nicht als einzelne Maschinen betrieben, sondern in sog. Server-Clustern mit u.U. einer Vielzahl an parallelen Maschinen. Fällt eine Maschine aus, dann übernimmt eine andere Maschine des Clusters deren Aufgaben. Durch entsprechende Protokolle bleibt dies dem Client verborgen.
Der in diesem Abschnitt präsentierte Fehlertoleranzmechanismus kommt erst zum Tragen, wenn diese Verfahren nicht zum Einsatz kommen bzw. ein Server trotz dieser Verfahren nicht erreicht werden kann.

Cache-basierte Verfahren unterliegen im Allgemeinen Einschränkungen, den Content betreffend: So ist es z.B. nicht sinnvoll, einen dynamischen Content in Caches zu speichern, der in Realzeit von einem originären Server zur Verfügung gestellt wird. Dies sind sich u.U. stark ändernde Informationen, wie dies z.B. bei Börsenkursen oder Online-Auktionen der Fall ist. Auch personalisierter Content sollte nicht in einem Cache gespeichert werden.
Des Weiteren ist es ebenfalls nicht sinnvoll bzw. unmöglich, Daten, die über eine verschlüsselte Kommunikationsverbindung zwischen Client und Server ausgetauscht werden, in einem Cache bereitzustellen.
Um derartige Anwendungen zu schützen, sind die beschriebenen CDN – eventuell in Kombination mit fehlertoleranten Endsystemen (z.B. FT-TCP) – die bessere Wahl. Das in Abschnitt 5.3 beschriebene Ersatzschalten durch Overlay-Netze ist jedoch eine gute Ergänzung von derartigen Architekturen.

Der in diesem Abschnitt präsentierte Ansatz bezieht sich somit nur auf Content, der sich *nicht häufig* ändert und ohne Zugangskontrolle (Personalisierung bzw. Verschlüsselung) auf einem originären Server zur Verfügung gestellt wird. Der durch unsere Architektur unterstützte Content kann sich ändern bzw. es kann dynamisch neuer Content zur Verfügung gestellt werden. Dies geschieht jedoch nicht in Realzeit, sondern in zeitlichen Intervallen, die mindestens in der Größenordnung einiger Stunden liegen.
Für eine Vielzahl von im Internet erhältlicher Information trifft diese Kategorie zu. Dies können z.B. Informationen sein, wie sie in der Regel auf Servern im universitären Umfeld zur Verfügung gestellt werden. Auch populäre Internetseiten, die z.B. Tageszeitungen archiviert darstellen, fallen in diese Kategorie. Die in letzter Zeit immer beliebter gewordenen privaten Homepages, gehören in der Regel ebenfalls dazu.

Im Folgenden beschreiben wir die notwendigen Ergänzungen, zu der in den Abschnitten 5.2.1 und 5.3 vorgeschlagenen Architektur kooperierender Overlay-Netze. Wir betrachten im Wesentlichen zwei Fälle:

- Das LAN, in dem der Server konnektiert ist, kann nicht erreicht werden. Somit sind sowohl der Server als auch der entsprechende Zugangsknoten nicht zu erreichen.
- Der Server ist ausgefallen. Das LAN, in dem der Server konnektiert ist, ist aber erreichbar. Auch der Zugangsknoten dieses LANs funktioniert fehlerfrei.

Auf den Fall, dass nicht nur der Server nicht zu erreichen ist sondern auch das LAN, in dem der Server konnektiert ist, gehen wir im Abschnitt 5.4.1 ein. Der Server kann, muss aber dabei nicht ausgefallen sein. Durch den Ausfall des Zielnetzes sind weder Server noch der entsprechende Zugangsknoten zu erreichen.
In Abschnitt 5.4.2 gehen wir auch darauf ein, dass nur der Server ausgefallen ist. Der dazugehörende Zugangsknoten ist aber erreichbar und arbeitet fehlerfrei.

5.4.1 Overlay-Netze von kooperierenden Cache-Clustern

In diesem Abschnitt gehen wir auf den Fall ein, dass, im Rahmen einer Client-Server-Kommunikation, das LAN in dem der Server konnektiert ist, nicht zu erreichen ist. Weiter gehen wir davon aus, dass auch der in Abschnitt 5.2.1 beschriebene Zugangsknoten, der den Server mit dem Internet konnektiert, nicht zu erreichen ist.
Für diesen Fall beschreiben wir die Funktionsweise des in Abbildung 5.1 angeführten Moduls zur Findung und Bereitstellung von alternativem Content, und integrieren die in [BPV+03] vorgeschlagene Architektur kooperierender Caches in die Dienstarchitektur.

Ziel ist zum einen, die domänenbegrenzte Kooperation aller Zugangsknoten bzw. deren Caches. Ähnlich zum im Abbildung 5.5 dargestellten Modell der dynamischen RON, schlagen wir auch hier einen dynamischen Zusammenschluss kooperierender Caches vor. Zum anderen berücksichtigen wir die Integration von Content-Distribution-Netzen (CDN siehe Abschnitt 3.3.2).
Grundsätzlich gehen wir von folgenden Erweiterungen in den Zugangsknoten aus:

- Jeder Zugangsknoten verfügt über einen lokalen Cache.
- Von seinem lokalen Cache erstellt ein Zugangsknoten eine Zusammenfassung.

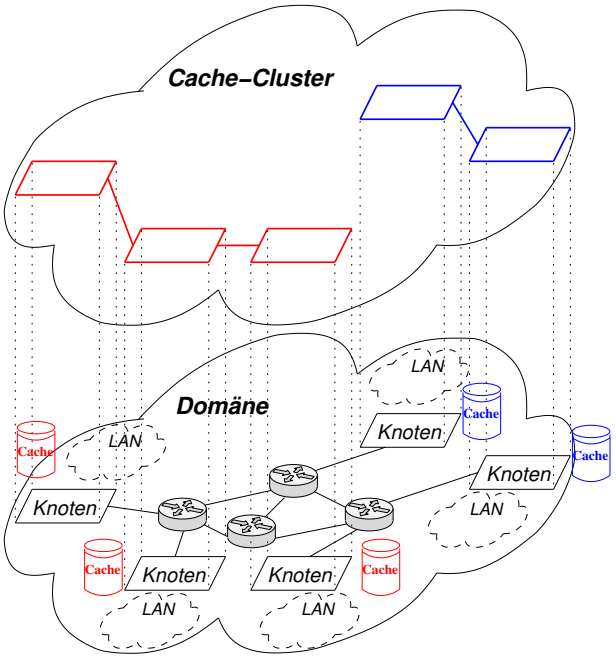

Abbildung 5.6: Kooperierende Cache-Cluster

- Jeder Zugangsknoten ist Teil eines Cache-Clusters.
- Cache-Cluster kooperieren in domänenbegrenzten Overlay-Netzen.
- Ein Zugangsknoten verfügt über die Information, ob Server im konnektierten LAN in einem CDN integriert sind.

In Abbildung 5.6 ist ein Überblick über die Architektur gegeben. Jeder Zugangsknoten innerhalb einer Domäne verfügt über eine Verbindung zu einem lokalen Cache. Dies kann entweder ein ohnehin im LAN vorhandener Cache sein, oder der Knoten arbeitet selbst als Cache.
Alle Zugangknoten berechnen für ihren Cache eine Cache-Summary (Zusammenfassung). Dies stellt eine komprimierte Information über die gespeicherten Dokumente im Cache dar. In einer administrativen Domäne gruppieren sich Zugangsknoten zu Cache-Clustern. Innerhalb eines jeden Cache-Clusters sind Zugangsknoten über einen Multicast-Baum miteinander verbunden. Über diesen Multicast-Baum tauschen alle Zugangsknoten eines Cache-Clusters ihre Cache-Summaries kontinuierlich miteinander aus. Jeder Zugangsknoten eines Cache-Clusters verbindet sich mit Zugangsknoten in allen anderen Cache-Clustern seiner administrativen Domäne. Somit entstehen Overlay-Netze, die jeweils Zugangsknoten aus allen unterschiedlichen Cache-Clustern miteinander verbinden (siehe Anhang A).

Als Referenzgrundlage nehmen wir den im Leibniz-Rechenzentrum (LRZ) der Münchener Universitäten installierten Cache. Dieser Cache hat durchschnittlich ca. 20 Millionen Dokumente gespeichert [BPV+03]. Erstellt man für einen Cache dieser Größenordnung eine Zusammenfassung (Cache-Summary) so ergibt dies bei der Verwendung eines Bloom-Filters (mit einer False-Positive Wahrscheinlichkeit von kleiner 2%) ein File von ca. 20 MByte [BM02].
Nehmen wir eine maximale Größe des Cache-Clusters von 10 teilnehmenden Zugangsknoten

an, müssen innerhalb des Cache-Clusters in etwa 1,6 GBit an Daten (Cache-Summaries) über einen Multicast-Baum ausgetauscht werden. Verteilt man diese Daten über den Multicast-Baum eines Cache-Clusters in einem Intervall von ca. einer Stunde, ergibt sich eine durchschnittlich benötigte Übertragungsrate von ca. 500 KBit/s.
Durch diese Verteilung der einzelnen Cache-Summaries kann jeder der in einem Cache beteiligten Zugangsknoten ohne weitere Signalisierung berechnen, ob ein bestimmtes Dokument in einem der Caches seines Clusters gespeichert ist. Die Zugangsknoten eines Cache-Clusters verfügen alle über die identische Information, abgesehen von etwaigen zeitlichen Verzögerungen.
Die in Abschnitt 5.3.9 getroffene Annahme von maximal 256 Zugangsknoten pro administrativer Domäne führt bei einer maximalen Cache-Cluster-Größe von 10 beteiligten Knoten zu einer maximalen Anzahl von 26 Cache-Clustern innerhalb einer Domäne.
Diese Cache-Cluster wählen jeweils einen Zugangsknoten aus, der als Cache-Cluster-Head fungiert. Diese Cache-Cluster-Heads konnektieren sich über ein eigenständiges, domänenbegrenztes Overlay-Netz.

Für den exemplarisch herangezogenen Cache des LRZ haben Messungen ergeben, dass im Durchschnitt etwa 30% der Anfragen durch den Cache bedient werden können, und somit die angefragten Daten nicht über das Internet übertragen werden müssen. Diese, durch den Einsatz von Caches mögliche, erhebliche Reduktion des Verkehrsaufkommens, ist durch eine Vielzahl an Messungen (siehe z.B. [AC98]) seit geraumer Zeit bekannt, und der Grund für die gegenwärtig hohe Verbreitung von Cache-Systemen im Internet. In dieser Arbeit bauen wir auf die *Summary-Cache*-Architektur von Li Fan [FCAB00] auf, und erweitern die Skalierbarkeit durch die Clusterung in Kombination mit einem Overlay-Netz. Besonderes Augenmerk liegt dabei auf dem (im Vergleich mit Summary-Cache) angestrebten, geringen Update-Intervall von ca. einer Stunde.

Gehen wir weiter von den in Abschnitt 5.3.9 getroffenen Annahmen aus, werden pro Zugangsknoten in der Sekunde in etwa vier Anfragen nach Content gestellt. Bei einer angenommenen Trefferrate von 30% kann im Durchschnitt mindestens eine Anfrage pro Sekunde durch den lokalen Cache bedient werden. Drei Anfragen pro Sekunde bleiben außen vor.
In einem nächsten Schritt wird versucht, die Anfragen innerhalb des eigenen Cache-Clusters zu bedienen. Laut Li Fan ergibt sich bei 10 kooperierenden Caches im Mittel eine Verdoppelung der Trefferrate für jeden Cache. Somit können wir davon ausgehen, dass in etwa die Hälfte aller Anfragen eines Zugangsknotens durch lokale Caches innerhalb eines jeden Cache-Cluster abgefangen werden können.
Jeder Zugangsknoten eines Cache-Clusters ist Teilnehmer eines Overlay-Netzes, das ihn mit Zugangsknoten in anderen Cache-Clustern innerhalb seiner administrativen Domäne verbindet. Daraus ergibt sich pro Zugangsknoten ein zu erwartendes Datenaufkommen für die Signalisierung mit anderen Cache-Clustern von ca. 1,2 KBit/s, inklusive Overhead (siehe Anhang A).
Mit diesem geringen Aufwand für die Signalisierung können, in Kombination mit dem Austausch innerhalb der Cache-Cluster, alle Caches in einer administrativen Domäne berücksichtigt werden. Das Resultat ist eine weitere Erhöhung der Trefferrate. Interpoliert man die Messungen von Li Fan, so ist ein Ansteigen der möglichen Trefferrate auf ca. 60-70 % für alle Anfragen zu erwarten.

5.4.2 Integrierte CDN

Die in Abschnitt 5.4.1 beschriebenen, kooperierenden Caches werden herangezogen, wenn nicht nur der Server sondern auch das entsprechende LAN mit Zugangsknoten nicht zu erreichen ist.
In diesem Abschnitt gehen wir auf den Fall ein, dass ausschließlich der Server im Rahmen einer Client-Server-Kommunikation ausfällt bzw. nicht zu erreichen ist.
Wesentlich für diesen Abschnitt ist die Voraussetzung, dass der Zugangsknoten, der das LAN, in dem der betreffende Server konnektiert ist, mit dem Internet verbindet, fehlerfrei funktioniert. Des Weiteren gehen wir davon aus, dass diesem Zugangsknoten folgende Informationen über den entsprechenden Server bzw. den darauf allozierten Content zur Verfügung stehen:

- Ist der ausgefallene Server ein Teilnehmer in einem *regulären* CDN, so muss diese Information dem Zugangsknoten zur Verfügung stehen.
- Der Zugangsknoten des Servers fungiert als *Inbound-Cache* und hat somit u.U. Content des Servers gespeichert.

Ist der ausgefallene Server ein Teilnehmer in einem CDN, wie in Abschnitt 3.3.2 beschrieben, muss dem Zugangsknoten des Servers dies bekannt gemacht werden. Im Falle einer Anfrage liefert der serverseitige Zugangsknoten diese Information an den clientseitigen Zugangsknoten. Dieser kann dann, wie dies in CDN üblich ist, als dynamischer Umleitungs-Server (engl. Redirection Server) [TG04] agieren und z.B. einen *HTTP-Redirect* an den Client senden mit der Adresse eines Replika-Servers für den ausgefallenen Server. Auf diese Weise ist es möglich, vorhandene CDN für die in Abschnitt 4.2.2 beschriebene longitudinale Fehlertoleranz effizient zu benützen. Zugangsknoten werden durch dieses Vorgehen kaum belastet, da der Content ausschließlich durch Server des entsprechenden CDN geliefert wird.

Ist ein Server nicht Teil eines CDN, stellt der Fehlertoleranzdienst selbst eine Art Replika-Server, und somit ein improvisiertes CDN, zur Verfügung. Der Zugangsknoten, der den Server an das Internet konnektiert, betreibt ein Inbound-Cacheing. Damit ist gemeint, dass der Zugangsknoten Dokumente, die aus dem Internet vom Server angefragt werden, kopiert und speichert, wenn sie diesen beim Transfer zwischen Server und Client passieren.
Dadurch sind im Cache des Zugangsknotens diejenigen Dokumente des Servers gespeichert, die vor dem Ausfall des Servers aus dem Internet angefragt wurden. Mit einer gewissen Wahrscheinlichkeit, abhängig von der tatsächlichen Benutzung des entsprechenden Servers, ist somit ein bestimmtes Dokument im Cache des Zugangsknotens gespeichert. Bei einem Ausfall des Servers übernimmt der serverseitige Zugangsknoten die Rolle eines Replika-Servers und stellt – falls vorhanden – angefragte Dokumente zur Verfügung.

5.4.3 Contentfindung

In den Abschnitten 5.4.1 und 5.4.2 sind verschiedene Methoden der longitudinalen Fehlermaskierung des in dieser Arbeit vorgeschlagenen Fehlertoleranzdienstes beschrieben.
Grundsätzlich werden CDN und Cache-Systeme gegenwärtig aus anderen Gründen im Internet eingesetzt. Das Augenmerk liegt nicht auf der Verbesserung der Fehlertoleranz, sondern

zum einen auf der Reduzierung des insgesamt durch das Internet übertragenen Datenaufkommens. Zum anderen kann auch eine Verkürzung der Laufzeiten und somit eine Verbesserung der Qualität für den Benutzer erreicht werden.

In dieser Arbeit schlagen wir einen anderen Ansatz vor. Wir nutzen die beschriebenen Mechanismen ausschließlich zur Maskierung von Applikationsfehlern. So benutzt der beschriebene Dienst Caches bzw. CDN nur, wenn nötig.
Für den Betrieb des Fehlertoleranzdienstes ist es notwendig, die Methoden im Rahmen einer Ablaufbeschreibung (engl. *Policy*) zu integrieren.
Betrachtet man aus Sicht des clientseitigen Zugangsknotens (im Rahmen einer Client-Server-Kommunikation) die Aspekte der Fehlertoleranz, hat der Fehlertoleranzdienst für den Fall der longitudinalen Maskierung folgenden Ablauf:

1. Content wird angefragt.
2. Originärer Server ist nicht zu erreichen, der entsprechende Zugangsknoten aber schon:
   - Holen des Contents aus Cache im Zugangsknoten, oder aus CDN des Servers, falls vorhanden.
3. Originaler Server und Zugangsknoten sind nicht zu erreichen:
   - Holen des Contents aus Caches in eigener Domäne.

Laut einer kürzlich veröffentlichten Studie [Dam04] beträgt die Verfügbarkeit (engl. Availability) von ans Internet angeschlossenen Servern derzeit ca. 99,3 %. Bei sog. hochzuverlässigen Kommunikationssystemen [Chr02], wie z.B. dem herkömmlichen Telefonnetz, beträgt die Verfügbarkeit im Vergleich dazu 99,999 %. Somit sehen wir prinzipiell noch einen erheblichen Aufholbedarf der internetbasierten Kommunikation, die Verfügbarkeit betreffend.
Die gemessene Verfügbarkeit von 99,3 % der ans Internet angeschlossenen Server bedeutet somit auf einen clientseitigen Zugangsknoten abgebildet, mit der getroffenen Annahme von ca. vier neuen Verbindungen pro Sekunde, dass in etwa 100 Verbindungen, ausgehend von Clients im LAN des Zugangsknoten, zu Servern pro Stunde nicht möglich sind aufgrund von Ausfällen der Server. Für diese ca. *100 Verbindungen pro Stunde* greift die beschriebene longitudinale Fehlermaskierung und versucht den angefragten Content von anderen Orten im Netz zu beziehen.
Wichtig ist aus der Sichtweise der Skalierbarkeit, die absolut gesehen geringe Zahl der zu erwartenden Contentausfälle pro Zeit. Wird durch den Fehlertoleranzdienst ein alternativer Content gefunden, muss dieser vom gefundenen, alternativen Cache im Netz zum entsprechenden Zugangsknoten bzw. Client, auch transportiert werden. Der alternative Cache agiert in diesem Fall als dynamischer Replika-Server eines CDN. Aufgrund der zu erwartenden, geringen Anzahl von tatsächlich bei entsprechenden Caches angefragten alternativen Contents, ist für den tatsächlichen Datentransport jedoch kein Skalierbarkeitsproblem zu erwarten.

Die beschriebene Architektur kann als Erweiterung, auch in ihrem ursprünglichen Sinn, als ein System kooperierender Caches wie dies Li Fan vorschlägt, genutzt werden. Für diesen Fall wären die entsprechenden Caches stärker frequentiert und müssten entsprechend leistungsfähiger ausgelegt werden.

Dies ist nicht der Fokus dieser Arbeit. Wir sehen darin lediglich eine mögliche Erweiterung bzw. Ergänzung des vorgeschlagenen Fehlertoleranzdienstes.

Zusätzlich zu den beschriebenen Verfahren ist durch den beschriebenen Dienst auch eine vertikale Fehlermaskierung möglich. Dies wird direkt von den Endsystemen initiiert. Im Wesentlichen geschieht dies durch eine dynamische Anpassung der flow-spezifischen Module. Zusätzlich zu den präsentierten Maskierungen werden Datenströme vom Fehlertoleranzdienst dann auch noch an die Gegebenheiten der Endsysteme adaptiert, wie z.B. in [KCD+00] beschrieben.

5.5 Ein neuer Datentransportmechanismus für Overlay-Netze

In diesem Abschnitt präsentieren wir einen neuen Mechanismus, der Daten in einem Overlay-Netz transportiert. Im Vergleich zu den in Abschnitt 3.4 präsentierten Verfahren hat das in diesem Abschnitt präsentierte *Sparse Networkspanning IP Label Switching* (SNIPLS) [BTK03] folgende Vorteile:

- Transparenter Betrieb des Overlay-Netzes gegenüber den Endsystemen ist möglich.
- Im Vergleich zu den bekannten Verfahren wird in SNIPLS weniger Overhead beim Transport benötigt.

Eine Gemeinsamkeit der beschriebenen Verfahren ist der Datentransport in einem Overlay-Netz durch die Einführung eines zusätzlichen Headers. Basierend auf diesem Header, werden Datenpakete in den beschriebenen Overlay-Netzen vermittelt.
Durch einen zusätzlichen Header werden Datenpakete bei der Aufnahme in das Overlay-Netz vergrößert. Abhängig vom Einzelfall, kann es daher vorkommen, dass die zulässige Gesamtgröße für Pakete überschritten wird. Diese Datenpakete können dann ohne weiteren Eingriff nicht über das Internet transportiert werden. Wie in Abschnitt 3.4.2 angeführt, wird dies durch eine Anpassung in den Endsystemen geregelt. Somit können Overlay-Netze nicht transparent an Endsysteme angeschlossen werden.

Der durch Enkapsulation eingeführte Overhead nimmt bei kleinen Datenpaketen, wie z.B. den häufig vorkommenden *TCP-Acknowledgements*, u.U. einen prozentual großen Anteil am Gesamtpaket ein. Die veröffentlichten Verfahren sind in diesem Punkt wenig effizient.

Das in diesem Abschnitt beschriebene allgemeine Verfahren für Datentransport in Overlay-Netzen führt keine zusätzlichen Header ein. Somit wird die Größe der einzelnen Datenpakete nicht verändert.
Aus diesem Grund können Overlay-Netze transparent an Endsysteme angebunden werden und sind effizienter, was die Menge der insgesamt transportierten Daten betrifft.

Im Folgenden führen wir zunächst das Konzept des **virtuellen Labels** ein. Basierend auf virtuellen Labeln, stellen wir exemplarisch den Transport eines Datenpakets durch ein Overlay-Netz vor.

5.5.1 Virtuelle Label

Die grundsätzliche Idee der virtuellen Label ist die Benutzung bereits vorhandener Felder in Datenpaketen. Diese Felder sind Bausteine eines *virtuellen Labels*. Durch die Manipulation dieser Felder werden Datenpakete, innerhalb eines Overlay-Netzes, von einem Knoten zum anderen vermittelt.
Um Datenpakete umzuleiten, wird die IP-Zieladresse in einem IP-Paket auf die IP-Adresse des nächsten Knoten in der angestrebten Route des Overlay-Netzes geändert. Weitere Felder im IP-Paket, die den Flow eindeutig identifizieren, dienen ebenfalls als Teil des virtuellen Labels, und werden gegebenenfalls manipuliert.

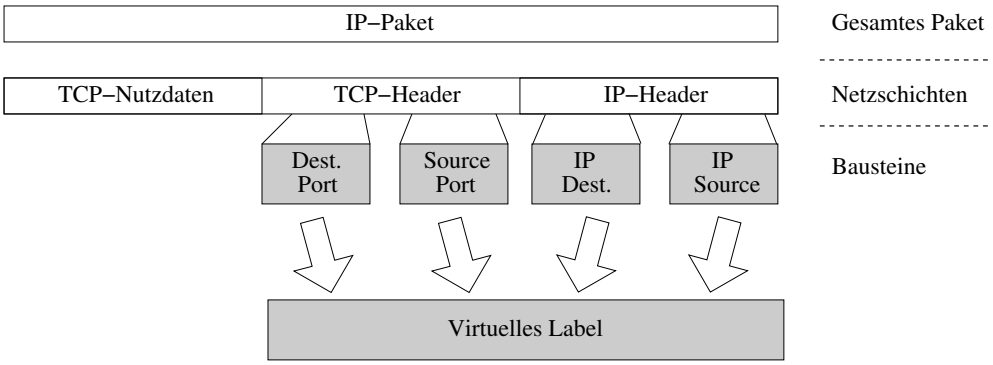

Abbildung 5.7: Virtuelles Label am Beispiel eines TCP/IP-Datenpakets

Am Beispiel eines TCP/IP-Datenpakets beschreiben wir im Folgenden exemplarisch ein virtuelles Label. In Abbildung 5.7 ist ein TCP/IP-Datenpaket nach den relevanten Feldern aufgeschlüsselt. Das Datenpaket unterteilt sich grob in IP-Header, TCP-Header und TCP-Nutzdaten. Innerhalb der Header gibt es verschiedene konstante Felder, die einen TCP-Datenfluss (engl. flow) zwischen zwei Endsystemen eindeutig charakterisieren. Aus diesen Feldern setzen wir virtuelle Label zusammen. Im Falle TCP/IP kann man z.B. aus IP-Quelladresse, IP-Zieladresse, TCP-Quellport und TCP-Sourceport ein virtuelles Label konstruieren.
Die IP-Zieladresse und die IP-Quelladresse werden jeweils auf die IP-Adressen der beteiligten Knoten im Overlay-Netz geändert. Die beiden TCP-Portnummern werden zur Identifizierung der entsprechenden Flows verwandt. Somit kann jeder einzelne Knoten eines Overlay-Netzes theoretisch maximal ca. $4,2*10^9$ unterschiedliche TCP-Datenflüsse* parallel verarbeiten und entsprechend weitertransportieren. Für das in Abschnitt 5.3 vorgestellte Modul des Fehlertoleranzdienstes, das Daten auf alternativen Wegen in einem Overlay-Netz transportiert, ist das bei Weitem ausreichend: Die in Abschnitt 5.3.9 behandelten Skalierbarkeitsbetrachtungen gehen lediglich von einigen hundert bis tausend Verbindungen über einen Zugangsknoten zur gleichen Zeit aus.

Das präsentierte Verfahren ist abhängig von den verwendeten Protokollen, und muss sich im Einzelfall entsprechend anpassen. Soll z.B. ein UDP-Datenstrom im Overlay-Netz geschaltet werden, dann bestehen die virtuellen Label, analog zum Beispiel, aus den UDP-Portnummern.

*Jeweils Unterscheidung nach Portnummern: $2^{16+16} = 4,2 * 10^9$

5.5.2 Eine geschaltete Route im Overlay-Netz

In Abbildung 5.8 zeigen wir exemplarisch eine, in einem Overlay-Netz geschaltete Route für eine Kommunikationsverbindung zwischen einem Client und einem Server. Sowohl auf der Seite des Client, als auch auf der Seite des Servers, ist jeweils ein Zugangsknoten (siehe Abbildung 5.1) im Datenpfad installiert. Ein weiterer Knoten des Overlay-Netzes ist durch den in Abschnitt 5.3.8 beschriebenen Mechanismus bereits gefunden, und stellt einen alternativen Pfad für die Verbindung zwischen Client und Server durch das Internet bereit.
Die beteiligten Knoten sind bereits entsprechend konfiguriert. Das bedeutet, die virtuellen Label sind für den entsprechenden Datenstrom bereits festgelegt. Jeder der beteiligten Kno-

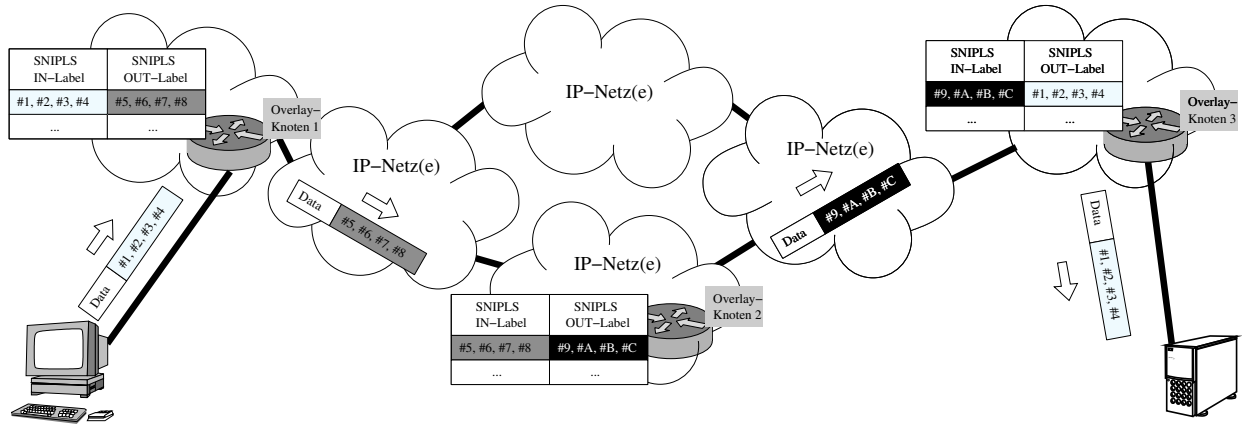

Abbildung 5.8: Eine im Overlay-Netz geschaltete Route für eine TCP-Verbindung

ten verfügt über die Information, wie er empfangene Datenpakete, die ein entsprechendes, virtuelles Label haben (IN-Label), manipulieren muss (d.h. das OUT-Label setzen), wenn er sie vermittelt.

Im Beispiel wird ein Datenpaket vom Client mit dem virtuellen Label (*#1, #2, #3, #4*) versandt. Dieses ist an den Server adressiert. Der clientseitige Zugangsknoten empfängt dieses Datenpaket und ändert die virtuellen Label gemäß seinen Informationen ab. Dadurch wird das Datenpaket in das Overlay-Netz aufgenommen.
Schließlich erreicht das weiter veränderte Datenpaket den Zugangsknoten auf der Seite des Servers. Dort werden die Felder des virtuellen Labels wieder auf die ursprünglichen Werte zurückgesetzt. Damit verlässt das Datenpaket das Overlay-Netz und wird zum Server gesendet. Dieser erhält somit das ursprüngliche Datenpaket.

5.6 Zusammenfassung

In diesem Kapitel geben wir einen Überblick über die modulare Softwarearchitektur des Fehlertoleranzdienstes. Diese teilt sich grob in ein Framework und in flow-spezifische Module. Das Framework stellt die Verbindung für einen einzelnen Knoten zu Overlay-Netzen her, und beinhaltet die Verwaltung, Überwachung, Signalisierung, Steuerung und allgemeine Suchfunktionen des Dienstes.
Das Framework konfiguriert und unterstützt die jeweiligen flow-spezifischen Module. Diese stellen die eigentliche Fehlertoleranz bzw. Fehlermaskierung für die jeweils zu schützenden Instanzen von Applikationen zur Verfügung.

Die wesentliche Funktionalität des Fehlertoleranzdienstes beruht auf der Integration verschiedener, kooperierender Overlay-Netze im Framework. So werden mittels eines übergeordneten Overlay-Netzes alle vorhandenen Zugangsknoten in den Dienst integriert. Die beschriebene Skalierbarkeit in der Anzahl der Teilnehmer ist eine wesentliche Differenzierung dieser Arbeit zu den in Abschnitt 3.3 angeführten Ansätzen.
Die Skalierbarkeit beruht zum einen auf der Trennung von Signalisierung und Bereitstellung des Dienstes in Framework und flow-spezifischen Modulen.
Zum anderen integriert diese Arbeit die notwendige Signalisierung für horizontale und longitudinale Fehlermaskierung jeweils durch dedizierte skalierbare Overlay-Netze im Framework. Somit werden die in den Abschnitten 3.3.1 und 3.3.2 vorgeschlagenen Ansätze im beschriebenen Fehlertoleranzdienst kombiniert.

Durch das Framework werden flow-basierte Module konfiguriert, die den Fehlertoleranzdienst letztlich bereitstellen. Im Fall der horizontalen Fehlermaskierung stellen diese Module, an RON angelehnt, dynamische Overlay-Netze dar. Die longitudinale Fehlermaskierung wird ebenfalls durch kooperierende Module in Overlay-Netzen bereitgestellt. Zusätzlich können flow-basierte Module noch dynamisch um Methoden erweitert werden, die eine vertikale Fehlermaskierung erlauben.
Eine weitere Neuerung, im Rahmen der Module, ist ein Mechanismus des Datentransports in Overlay-Netzen. Durch eingeführte virtuelle Label ist es möglich, Datenverkehr in einem Overlay-Netz zu transportieren, ohne dass dazu ein Overhead (in Form zusätzlicher Header) notwendig ist. Daraus ergibt sich unmittelbar der Vorteil der Effizienz, da weniger Redundanz übertragen werden muss. Des Weiteren ist dieses Verfahren insbesondere für 1+1 Ersatzschaltverfahren in Overlay-Netzen geeignet, da es vollkommene Transparenz gegenüber beteiligten Endsystemen ermöglicht.

Kapitel 6

Eine komponentenbasierte programmierbare Knotenarchitektur

Der in Kapitel 5 beschriebene Fehlertoleranzdienst ist als ladbarer, adaptiver Dienst in Programmierbaren Netzen konzipiert. Grundlage der Programmierbaren Netze sind programmierbare Knotenarchitekturen (siehe Abschnitt 3.7).

Die in Abschnitt 5.3.9 getroffenen Annahmen über den Betrieb des Fehlertoleranzdienstes stellen Anforderungen an die Leistungsfähigkeit des Fehlertoleranzdienstes, und somit auch an die zugrunde liegende programmierbare Plattform.
Wir gehen von vier neuen Verbindungen pro Sekunde aus, die von Endsystemem über einen Zugangsknoten mit dem Internet aufgebaut werden. Bei einer angenommenen durchschnittlichen Dauer einer Verbindung von einer Minute ergeben sich ca. 250 Verbindungen, die von einem Zugangsknoten zeitgleich bearbeitet werden müssen. Da der Fehlertoleranzdienst jede einzelne Verbindung dediziert behandelt, entspricht die Anzahl der Verbindungen der parallel auf einem Zugangsknoten ablaufenden *flow-spezifischen* Module (siehe Abschnitt 5.1). Diese doch relativ große Anzahl an z.T. rechenintensiven Modulen muss von der zugrunde liegenden programmierbaren Plattform parallel zur Verfügung gestellt werden können.
In der Entwicklung der beschriebenen Ansätze programmierbarer Knoten wurde die lokale Skalierbarkeit, und somit Performanz eines Knotens, bisher nur am Rande betrachtet. In diesem Kapitel schlagen wir eine neue Architektur eines programmierbaren Knotens vor, der diesen Anforderungen des Fehlertoleranzdienstes entspricht.

Mehrrechner-Systeme (siehe Abschnitt 3.7.2) beschreiben erste Ansätze, die lokale Skalierbarkeit eines programmierbaren Knotens zur Verfügung stellen. Da diese Ansätze aber lediglich Kombinationen aus den in Abschnitt 3.7.1 dargelegten Einzelrechner-Systemen sind, haben sie ebenfalls die in Abschnitt 3.7.3 beschriebenen Probleme die Sicherheit und Zuverlässigkeit betreffend.

Die in diesem Kapitel beschriebene neue Architektur umgeht diese Problematik durch eine räumliche Trennung der grundlegenden Funktionalitäten eines programmierbaren Knotens:

- Routing
- Signalisierung bzw. Steuerung des Knotens.
- Bereitstellung der ladbaren Dienste.

Durch die Architektur können Dienste fehlertolerant und skalierbar bereitgestellt werden. Ein weiterer Vorteil der vorgeschlagenen Architektur ist die Verbesserung der Sicherheit des programmierbaren Knotens gegenüber Angriffen, die von fehlerhaftem bzw. "böswilligen"Code ausgehen, der u.U. auf den programmierbaren Knoten geladen werden könnte.

Im Folgenden beschreiben wir zunächst die vorgeschlagene Architektur und stellen die Komponenten in einem Überblick dar. Des Weiteren präsentieren wir die interne und externe Signalisierung des Knotens. Zum Abschluss gehen wir noch auf das Zusammenspiel des beschriebenen Fehlertoleranzdienstes mit der Knotenarchitektur ein.

6.1 Überblick über die Komponenten

Die Grundlage für die beschriebene Mehrrechner-Architektur ist die Aufteilung von Komponenten des programmierbaren Knotens (*Routing*, die *Bereitstellung von Diensten* und die *Signalisierung bzw. die Steuerung* des Knoten) auf unterschiedliche Maschinen.

In Abbildung 6.1 ist ein Überblick über die Architektur gegeben. Grundsätzlich unterteilen wir einen programmierbaren Knoten in eine Router-Komponente und eine Rechner-Komponente [BTS+03]. Kern der Architektur ist die Integration einer Enkapsulierung bzw.

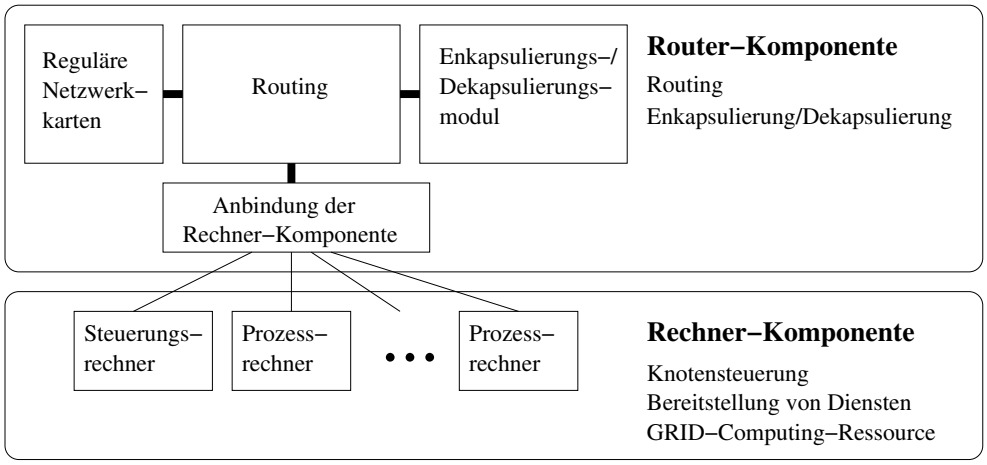

Abbildung 6.1: Komponentenbasierte Mehrrechner-Architektur eines programmierbaren Knotens

Dekapsulierung von Datenpaketen in der Router-Komponente. Durch diesen Mechanismus können IP-Datenpakete, die am programmierbaren Knoten eintreffen, von der Router-Komponente zu den entsprechenden Diensten weitergeleitet werden.

In der vorgeschlagenen Architektur laufen beschriebene Dienste ausschließlich innerhalb der Rechner-Komponente auf dedizierten Prozessrechnern im *Userspace* ab. Durch die Enkapsulierung ist es möglich, dass auf den Prozessmaschinen die in Abschnitt 3.7.1 beschriebene Kommunikation zwischen Kern und Userspace umgangen werden kann. Die Komponenten des programmierbaren Knotens, auf denen ladbare Dienste ausgeführt werden, benötigen somit keinen zusätzlichen (vom Standard abweichenden) Mechanismus, der Datenpakete aus dem Kern des Betriebssystems in den Userspace kopiert.

Die einzig benötigte Schnittstelle für Dienste, um Datenpakete zu empfangen, sind Standard-UDP-Sockets. Somit ist es möglich, dass für die in Abschnitt 3.7.1 beschriebenen Ansätze übliche *Execution-Environment (EE)* zu umgehen. Die vorgeschlagenen Ansätze führen diese zusätzliche Softwareschicht ein. Die Aufgabe eines EE ist in erster Linie die Kommunikation mit dem Kern des Betriebssystems. Außerdem dient das EE als Programmbasis, zu der entsprechende Dienste dynamisch gelinkt werden können.
In dem Vorschlag dieser Arbeit wird auf ein dediziertes EE verzichtet. Die grundlegenden Funktionen eines EE werden direkt vom Betriebssystem übernommen (siehe [BTM+04]). Spezielle Erweiterungen, z.B. die Überwachung von Diensten, werden durch eigenständige Programme übernommen, die ebenfalls auf Standardfunktionen des verwendeten Betriebssystems aufbauen. Einer der Vorteile liegt darin, dass die in Abschnitt 3.7.3 beschriebenen Probleme der Sicherheit damit umgangen werden können.
Des Weiteren ist eine programmierbare Knotenarchitektur, ohne dediziertes EE, in der Praxis flexibler. Die Dienste können z.B. in einer Vielzahl an Programmiersprachen angeboten werden. Wichtig ist in diesem Zusammenhang nur die jeweilige Unterstützung durch das Betriebssystem; somit muss nicht für jede Sprache ein eigenes EE zur Verfügung stehen.

Die dritte wesentliche Komponente des programmierbaren Knotens ist die Kapselung der Signalisierung und Steuerung des Knotens durch dedizierte Signalisierungsrechner innerhalb der Rechner-Komponente.
Zum einen wird dadurch die Sicherheit des Gesamtsystems verbessert. Fehlerhafter oder bösartiger Code, der als Dienst geladen wird, hat somit keine direkte Möglichkeit die Steuerung des Knotens, z.B. durch einen provozierten Systemabsturz, zum Stillstand zu bringen. Zum anderen ist es durch die vorgeschlagene Architektur der Signalisierung möglich, Dienste fehlertolerant anzubieten. Fällt z.B. eine Maschine aus, die Instanzen von Diensten zur Verfügung stellt, kann eine weitere Maschine übernehmen und die Instanzen fortführen. Dadurch ist es möglich, programmierbare Dienste fehlertolerant anzubieten. Darüber hinaus lässt sich auch eine Lastbalancierung realisieren.

Basierend auf den drei beschriebenen Mechanismen präsentieren wir im Folgenden eine robuste Architektur eines komponentenbasierten, programmierbaren Knotens, die die in Abschnitt 3.7 präsentierten Ansätze, hinsichtlich der Sicherheit und Ausfallsicherheit der Systeme, verbessert.

6.1.1 Router-Komponente

In diesem Abschnitt gehen wir auf die Router-Komponente des programmierbaren Knotens ein. In Abbildung 6.2 ist das Zusammenspiel zwischen Router-Komponente und einem Prozessrechner innerhalb der Rechner-Komponente exemplarisch beschrieben.

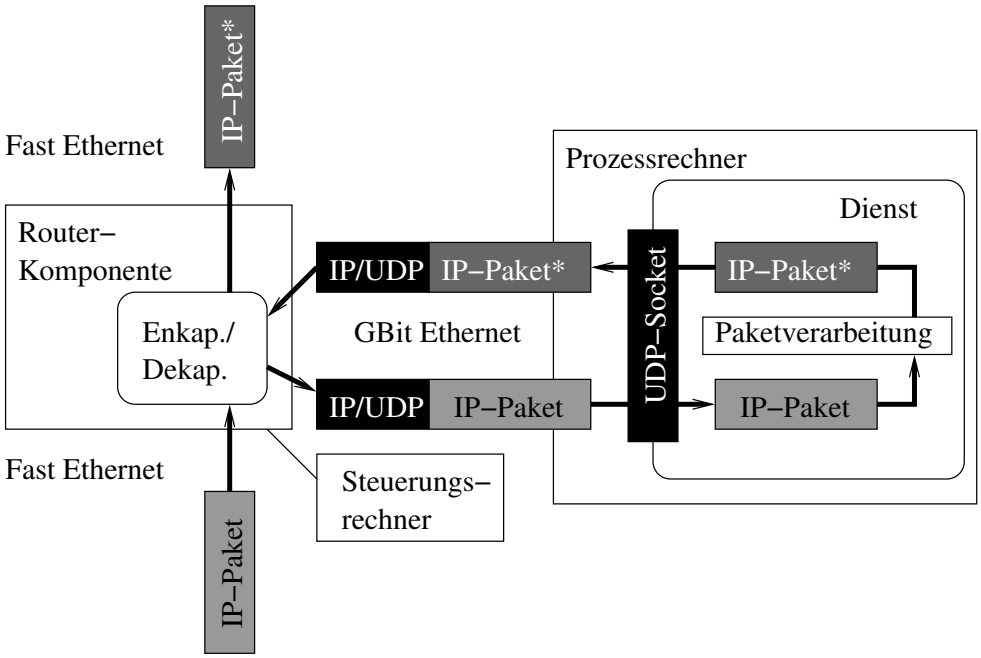

Abbildung 6.2: Exemplarischer Paketfluss durch den programmierbaren Knoten

Ein IP-Paket eines Datenstroms trifft am programmierbaren Knoten ein. Für diesen Datenstrom ist auf dem programmierbaren Knoten bereits ein Dienst gestartet. Innerhalb der Router-Komponente wird das IP-Paket in einem *Enkapsulierungs-/ Dekapsulierungs-Modul* mit einem zusätzlichen Header versehen. Dieser Header setzt sich zusammen aus einem IP-Header und einem UDP-Header*. Basierend auf diesem Header, wird das Datenpaket an einen entsprechenden Prozessrechner mit einem zum Datenpaket korrespondierenden Dienst weitergereicht. Dort wird das Datenpaket verarbeitet und wieder an die Router-Komponente zurückgeschickt. Die Router-Komponente dekapsuliert das Datenpaket. Im Anschluss wird das Datenpaket geroutet und verlässt den programmierbaren Knoten.

Ziel der Architektur ist es, eine kleine Erweiterung an einem Standard-Router vorzunehmen, und dadurch die Router-Komponente des programmierbaren Knotens zu bilden.
Im Folgenden gehen wir auf die in Abbildung 6.3 skizzierte, vorgeschlagene Erweiterung eines Standard-Routers – *das Enkapsulierungs-/ Dekapsulierungs-Modul* – ein. Das Modul ist der Routing-Funktionalität in einem Router vorgeschaltet.
Jedes an der Router-Komponente ankommende IP-Paket wird zunächst überprüft, ob es von der Rechner-Komponente des programmierbaren Knotens stammt. Ist dies der Fall, so wird überprüft, ob das Datenpaket dekapsuliert werden muss, d.h. der zusätzliche Header entfernt werden muss. Trifft dies zu, wird das Paket dekapsuliert. Im Anschluss wird das IP-Paket regulär geroutet und verlässt den programmierbaren Knoten.

Für den Fall, dass ein an der Router-Komponente ankommendes IP-Paket nicht von der Rechner-Komponente stammt, wird überprüft, ob das Datenpaket enkapsuliert werden muss. Dies ist der Fall, wenn das Datenpaket Teil eines Datenflusses ist, für den bereits schon ein Dienst auf dem programmierbaren Knoten gestartet worden ist.

*Fragmentierung von enkapsulierten Paketen tritt nicht auf. Zwischen den Komponenten wird ein GBit-Ethernet benutzt, das "Jumbo-Pakete"mit einer maximalen Größe bis zu 9 KByte erlaubt.

Des Weiteren werden Datenpakete enkapsuliert, die Teil eines neuen, bis dato unbekannten Datenflusses sind, der analysiert werden muss. Im Anschluss wird das IP-Paket regulär geroutet und an den entsprechenden Prozessrechner geschickt.

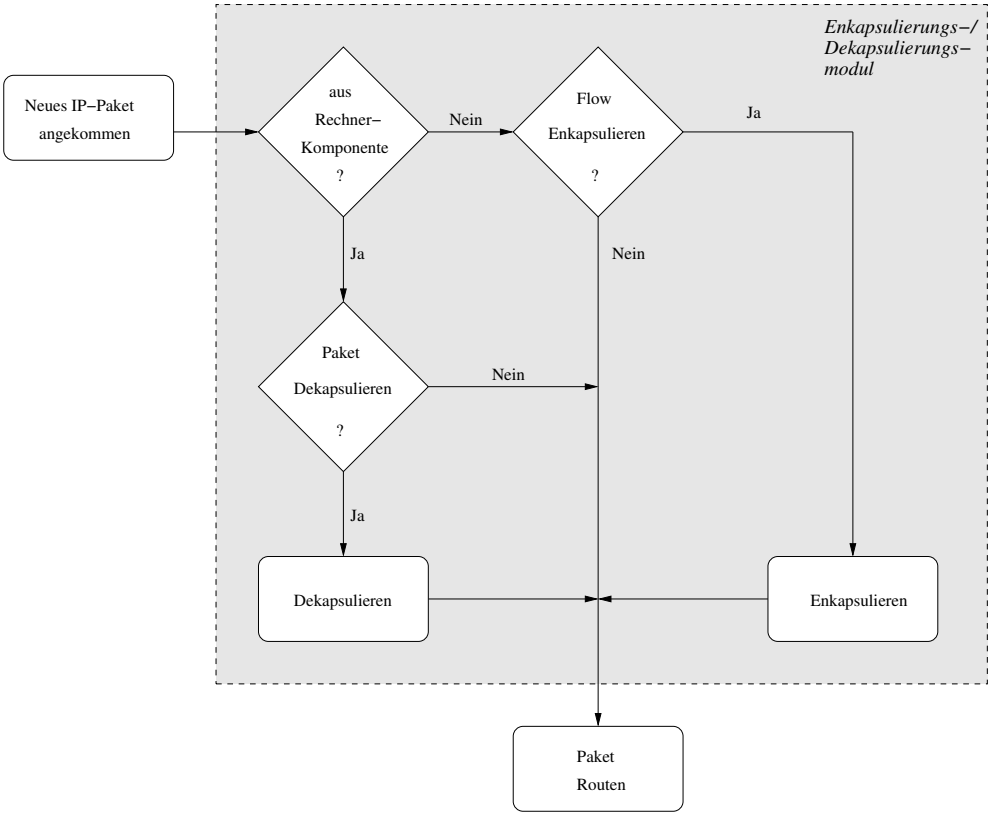

Abbildung 6.3: Das Enkapsulierungs-/ Dekapsulierungs-Modul in der Router-Komponente

Damit der in Abbildung 6.3 dargestellte Mechanismus funktionieren kann, ist eine lokale Datenhaltung in der Router-Komponente notwendig. Der Router-Komponente müssen Informationen über die gegenwärtigen, durch Dienste auf den Prozessrechnern bearbeiteten, Datenströme zur Verfügung stehen.
Der Umfang dieser benötigten Datenbank ist aber eher gering anzusehen. Gemäß den Annahmen sind pro Sekunde lediglich vier neue Einträge in die Datenbank zu erwarten. Insgesamt müssen nur Information über einige hundert Datenflüsse in der Datenbank gespeichert werden.
Zusätzlich ist noch eine Signalisierungsschnittstelle in der Router-Komponente notwendig. Diese verbindet die Router-Komponente mit den dedizierten Signalisierungsrechnern in der Rechner-Komponente. Durch diese Schnittstelle kann somit auf die Datenbank in der Router-Komponente zugegriffen werden.

Die grundsätzliche Idee der vorgeschlagenen Architektur ist es, einen Standard-Router durch das Enkapsulierungs-/ Dekapsulierungs-Modul zur Router-Komponente zu erweitern. An diese Router-Komponente erfolgt dann der in Abbildung 6.1 beschriebene Anschluss der Rechner-Komponente.
Die Komplexität des beschriebenen Moduls ist bewusst gering gehalten. Dadurch soll zum

einen der Einbau in einen Standard-Router vereinfacht bzw. ermöglicht werden; zum anderen ist der beschriebene Mechanismus schnell und beeinflusst kaum die Leistungsfähigkeit des Routers.
In einem Linux-basierten Prototyp wurde die Funktionalität des Moduls validiert [BTM+04]. Das zusätzliche Modul beeinflusste sowohl den gemessenen Durchsatz als auch die Verzögerung nur unwesentlich, im Vergleich zum Betrieb des Routers ohne Modul. Dies liegt in erster Linie daran, dass die Enkapsulierung und Dekapsulierung sehr schnell durchführbar sind. Im Wesentlichen müssen nur einige, wenige Byte kopiert bzw. gelöscht werden.
Die durch das Modul bedingte Verzögerung war im Prototyp nicht explizit messbar. Für Verzögerungen zeigten sich bei der prototypischen Implementierung in erster Linie Hardware-Limitierungen des verwendeten Standard-PC verantwortlich.

6.1.2 Rechner-Komponente

In diesem Abschnitt gehen wir auf die Rechner-Komponente des programmierbaren Knotens ein. Diese kann aus einer Vielzahl von regulären Rechnern bestehen, die an die Router-Komponente angeschlossen sind. Die einzelnen Rechner haben unterschiedliche, sich gegenseitig ausschließende, Aufgaben:

- Steuerung und Signalisierung des programmierbaren Knotens.
- Prozessrechner zur Bereitstellung der Dienste.

Steuerungs- und Signalisierungsrechner des programmierbaren Knotens stellen zum einen die Schnittstelle dar, um den programmierbaren Knoten von außen anzusteuern; zum anderen steuern diese Rechner die Router-Komponente und die Prozessrechner. In Abschnitt 6.2 präsentieren wir die Steuerung bzw. Signalisierung im Detail.

Im Folgenden betrachten wir die Prozessrechner der Rechner-Komponente. Aufgabe der Prozessrechner ist die Bereitstellung von Diensten. Dienste werden, wie bei den in Abschnitt 3.7 beschriebenen programmierbaren Knoten, auf die Prozessrechner geladen und dort ausgeführt. Diesen Vorgang initiieren die Steuerungs- und Signalisierungsrechner.
Dienste werden auf den Prozessrechnern durch reguläre Userspace-Programme zur Verfügung gestellt. Das in Abbildung 6.2 dargestellte Modul der Router-Komponente enkapsuliert ankommende Datenpakete mit einem IP/UDP-Header. Ein Dienst empfängt diese enkapsulierten Datenpakete über einen regulären UDP-Socket. Die somit empfangene Payload des enkapsulierten Datenpaketes entspricht dem ursprünglichen Datenpaket, inklusive des ursprünglichen IP-Headers. Der Dienst kann dann das ursprüngliche Datenpaket verarbeiten. Anschließend versendet der Dienst das bearbeitete Datenpaket wieder über den zur Verfügung stehenden UDP-Socket. Dies entspricht einer erneuten Enkapsulierung. Das enkapsulierte Datenpaket wird anschließend in der Router-Komponente dekapsuliert und geroutet.

Somit benötigt unsere Architektur auf den Prozessrechnern keine Änderung im Kern des Betriebssystem, wie dies für alle in Abschnitt 3.7 beschriebenen Architekturen der Fall ist. Auf den Prozessrechnern laufen ausschließlich reguläre Programme im Userspace. Diese greifen auf Standardfunktionen (z.B. UDP-Sockets) des Betriebssystems zurück.
Auch die lokale Kontrolle der Dienste auf den Prozessrechnern basiert auf der Nutzung von

Standardfunktionen des Betriebssystems. Jeder Dienst wird z.B. unter einer eineindeutigen User-ID gestartet. Stellt die lokale Kontrolle des Prozessrechners fest, dass ein lokaler Benutzer (d.h. ein bestimmter Dienst) ein Problem verursacht, kann entsprechend reagiert werden.
Durch dieses Grundprinzip der Nutzung von Standardfunktionen wird unsere Architektur unabhängig von der tatsächlichen Implementierung im Betriebssystem. Das Einspielen von Sicherheits-Updates, die Teile des Betriebssystems verändern, ist daher vollkommen transparent möglich.
In den gegenwärtigen Architekturen von programmierbaren Knoten stellt dies ein nicht zu vernachlässigendes Problem dar, da u.U. die Änderungen im Kern des Betriebssystems ebenfalls angepasst werden müssen.
Abgesehen davon, sehen wir weitere Vorteile unserer Architektur gegenüber dem Stand der Technik:

- Dienste können in einer Vielzahl von Programmiersprachen angeboten werden.
- Komplexe Dienste, die spezielle Bibliotheken von Funktionen (z.B. Transkodierung) benötigen, können leicht realisiert werden.
- Auf den einzelnen Prozessrechnern können unterschiedliche Betriebssysteme laufen.

Durch die Bereitstellung der Dienste als eigenständige Userspace-Programme bietet unsere Architektur somit eine höhere Flexibilität gegenüber dem Stand der Technik. Die Sicherheit der Knoten-Architektur wird, trotz gewonnener Flexibilität, nicht negativ beeinflusst, da die in Abschnitt 3.7.3 beschriebenen Sicherheitsmechanismen in unsere Architektur integrierbar sind.

6.2 Ebenen der Signalisierung

Die in Abbildung 6.1 dargestellten Signalisierungsrechner innerhalb der Rechner-Komponente haben zwei wesentliche Aufgaben: Zum einen die Steuerung des programmierbaren Knotens, zum anderen stellen sie die Schnittstelle des programmierbaren Knotens nach außen bereit.
Die grundlegende Signalisierung bezeichnen wir als *Basissignalisierung* des programmierbaren Knotens. Durch die Basissignalisierung wird der Knoten gesteuert bzw. ist der Knoten von außen ansteuerbar. Im Detail können damit ladbare Dienste auf dem Knoten gestartet werden bzw. werden Dienste innerhalb des Knotens überwacht.
Für den Fall, dass die auf den Knoten geladenen Dienste atomaren Charakter haben (wie z.B. [BTÜS03]), ist keine weitere Signalisierung notwendig. Derartige Dienste werden durch die programmierbare Plattform gestartet, überwacht und beendet. Seitens des Dienstes erfolgt keine weitere Signalisierung.
Für den in dieser Arbeit beschriebenen Fehlertoleranzdienst ist aber seitens des Dienstes eine eigene, zusätzliche Signalisierung notwendig. Der beschriebene Dienst basiert darauf, dass im Netz verteilte Dienstmodule miteinander kooperieren und Informationen austauschen. Diese dienstspezifische Signalisierung betrachten wir in dieser Arbeit als Teil eines Dienstes, der unabhängig von der Basissignalisierung der programmierbaren Plattform ist.

Deshalb präsentieren wir in dieser Arbeit zwei Ebenen der Signalisierung: Zum einen die Basissignalisierung, die für die Steuerung des Knotens zuständig ist. Zum anderen die dienstspezifische Signalisierung, die für komplexe, nicht-atomare Dienste notwendig ist.

6.2.1 Steuerung des Knotens

Die Steuerungs- und Signalisierungsrechner sind das koordinierende Element des programmierbaren Knotens. In Abbildung 6.4 ist ein Überblick über die Signalisierung gegeben.

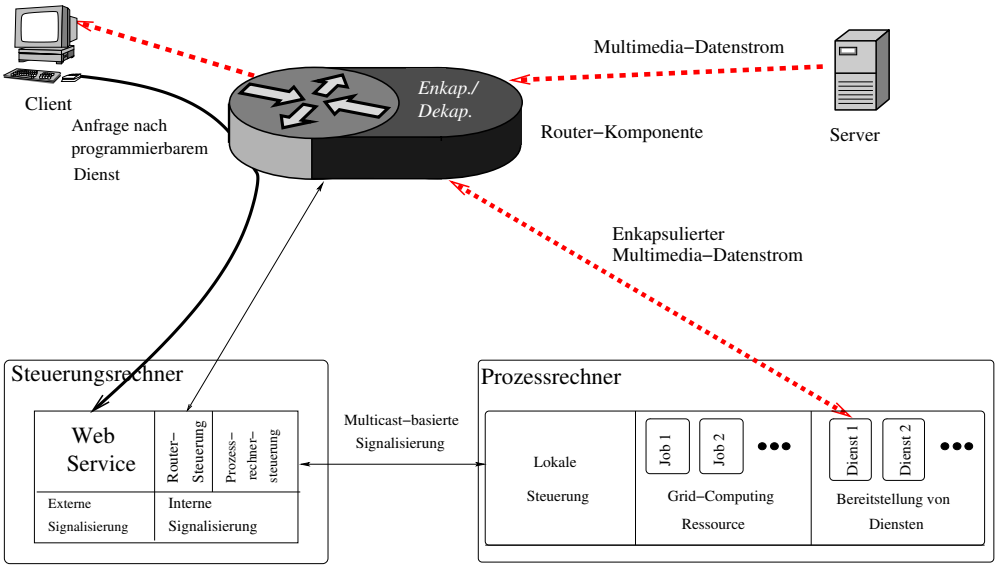

Abbildung 6.4: Einbettung der Signalisierung in die Knotenarchitektur

Um die Fehlertoleranz des Gesamtsystems zu erhöhen, sieht unsere Architektur einen Signalisierungs- bzw. Steuerungsrechner vor, dem parallel ein Rechner beigestellt ist, der im Falle eines Ausfalls die Signalisierung übernimmt. Diese Signalisierungs- und Steuerungsrechner haben folgende Aufgaben:

1. Schnittstelle nach außen.
2. Steuerung der Prozessrechner in der Rechner-Komponente.
3. Steuerung der Router-Komponente.

Diese Funktionen sind jeweils durch Software-Module auf den Signalisierungs- bzw. Steuerungsrechnern realisiert. Diese Module auf den Steuerungsrechnern starten und überwachen Dienste auf den Prozessrechnern. Sie synchronisieren die Router-Komponente mit den Diensten auf Prozessrechnern. Abgesehen von diesen internen Steuerungsfunktionen, stellen die Signalisierungs- bzw. Steuerungsrechner zusätzlich die Schnittstelle des programmierbaren Knotens nach außen dar.

Die externe Schnittstelle legt nur dienstunabhängige, allgemeine Funktionen offen. So können Dienste auf der programmierbaren Architektur gestartet werden. Bestimmte Parameter, wie z.B. die gegenwärtige Auslastung des Gesamtsystems bzw. eines speziellen Dienstes, können abgefragt werden.
Die externe Schnittstelle des programmierbaren Knotens wird als Web-Service [web04] zur Verfügung gestellt. Der implementierte Prototyp setzt dabei auf Java* auf und benutzt das *Web-Service-Framework*†.
Wir verwenden Web-Services als eine Schnittstelle, die es entfernten Applikationen ermöglicht, den programmierbaren Knoten anzusteuern. Dies könnte man auch mit einer proprietären Schnittstelle bewerkstelligen. Die Verwendung des Web-Service-Standards bringt jedoch Vorteile mit sich, wie z.B.:

- Authentifizierung, Zugang
- Abrechnung
- Einfache Erweiterung: Grid-Computing.

Durch Web-Services ist die Möglichkeit des Zugriffs auf andere, schon zur Verfügung stehende, Web-Services gegeben. So können Web-Services verwandt werden, die die Authentifizierung und betriebswirtschaftliche Aufgaben, wie die Abrechnung bei der Benutzung eines programmierbaren Knotens regeln.
Des Weiteren ist es möglich, die Ressourcen des programmierbaren Knotens in einem Grid-Computing-Netz über Web-Service zur Verfügung zu stellen. Darauf gehen wir in Abschnitt 6.2.3 detaillierter ein.

Die in Abschnitt 6.1.2 beschriebenen Prozessrechner werden ausschließlich durch die Signalisierungs- bzw. Steuerungsrechner angesteuert. Unsere Architektur sieht hier eine Kapselung vor. Auf die Prozessrechner ist kein direkter Zugriff von außen möglich.
Die durch Web-Service zur Verfügung gestellte Schnittstelle auf den Signalisierungsrechnern kommuniziert mit dem Software-Modul, das die Prozessrechner kontrolliert. Im Wesentlichen hat dieses Modul folgende Aufgaben:

- Überwachung der Prozessrechner.
- Lastbalancierung
- Fehlertolerante Bereitstellung von Diensten.
- Begleitung der Instanzen von Diensten durch deren Lebenszyklen.

Unsere Architektur sieht eine Vielzahl von beteiligten Prozessrechnern in der Rechner-Komponente vor. Diese Prozessrechner werden von den Steuerungsrechnern überwacht. Ziel ist zum einen eine Lastbalancierung, d.h. Dienste werden über die einzelnen Prozessrechner verteilt. In diesem Zusammenhang spielt insbesondere die gegenwärtige Auslastung der einzelnen Rechner eine Rolle. Darüber hinaus sollen Dienste fehlertolerant bereitgestellt werden. Somit muss ein Versagen eines einzelnen Dienstes bzw. auch eines Prozessrechner, ebenfalls überwacht werden.
Die Signalisierungs- bzw. Steuerungsrechner begleiten Instanzen von Diensten während ihres

*Java 1.3.1, Netbeans IDE 3.5
†JAX-RPC: Java API für XML-basierte RPC

Lebenszyklus. Das bedeutet: nach dem Start eines Dienstes auf einem Prozessrechner wird sowohl der Dienst als auch der entsprechende Rechner überwacht. Tritt ein Problem auf, das den weiteren Betrieb eines Dienstes auf einem Prozessrechner unmöglich macht, dann wird entsprechend reagiert. Dies kann im günstigsten Fall eine Verlagerung des entsprechenden Dienstes auf einen anderen Prozessrechner bedeuten. Ist dies nicht möglich, wird der entsprechende Dienst beendet.

Das in Abschnitt 6.1.1 beschriebene Modul der Router-Komponente wird ausschließlich durch die Signalisierungs- und Steuerungsrechner angesteuert. Aus Sicherheitsgründen ist es, sowohl von außen als auch von den Prozessrechnern, nicht direkt möglich die Zuordnung: Dienst $\Longleftrightarrow$ Datenstrom zu beeinflussen.
Diese Zuordnung kann in der Architektur nur durch die Router-Steuerung in den Signalisierungs- bzw. Steuerungsrechnern erfolgen. Beabsichtigt ein bestimmter Dienst diese Zuordnung zu ändern (weil z.B. ein bestimmter Datenstrom zusätzlich bearbeitet werden soll), dann teilt der Dienst dies dem Steuerungsrechner mit.

6.2.2 Dienstspezifische Signalisierung

In Abbildung 6.5 sind die zwei Ebenen der Signalisierung dargelegt. Die Basissignalisierung regelt die dienstunspezifischen Steuerungsfunktionen des programmierbaren Knotens. Für den Fall, dass nicht-atomare, komplexe Dienste auf der programmierbaren Plattform betrieben werden sollen, ist es notwendig, dass die Dienste – zusätzlich zur Basissignalisierung – dedizierte Signalisierungsprotokolle bedienen können.
Dies gilt z.B. für den in Kapitel 5 beschriebenen ladbaren Fehlertoleranzdienst. Der Dienst basiert auf im Netz verteilt betriebenen Einheiten. Diese Einheiten müssen miteinander kommunizieren, um z.B. die beschriebenen, kooperierenden Overlay-Netze aufzubauen. Auch für andere, nicht-atomare ladbare Dienste (wie z.B. Hot-TCP [TBW04]) wird eine zusätzliche, dienstspezifische Signalisierung vorausgesetzt.

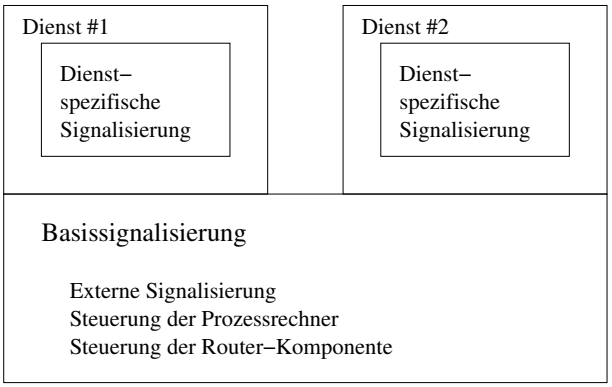

Abbildung 6.5: Die zwei Ebenen der Signalisierung

In Abbildung 6.5 stellen wir die zwei Ebenen der Signalisierung in einem laufenden System dar. Die Basissignalisierung wird von der programmierbaren Plattform betrieben und überwacht. Die dienstspezifische Signalisierung hingegen sehen wir als Teil des ladbaren

Dienstes an. In der vorgeschlagenen Architektur ist es daher nicht notwendig, dass die programmierbare Plattform direkt in die dienstspezifische Signalisierung eingreift, wie dies z.B. in [HSWZ01] vorgesehen ist. Dadurch wird unsere Architektur flexibler. Dienstspezifische Signalisierungsdaten müssen nicht zuerst mit der programmierbaren Plattform ausgetauscht werden, sondern werden direkt auf der Ebene der Dienste ausgetauscht.
Der programmierbare Knoten überwacht lediglich die dienstspezifische Signalisierung. Dies geschieht, wie in Abschnitt 6.1.2 beschrieben, durch die Nutzung von standardisierten Funktionen des Betriebssystems. So kontrolliert die Plattform z.B. für jede Instanz eines Dienstes die Anzahl der Verbindungen zur Signalisierung. Auch das transferierte Datenvolumen zur Signalisierung wird überwacht.

6.2.3 Konvergenzarchitektur für GRID-Computing

Die in diesem Kapitel präsentierte Architektur eines komponentenbasierten programmierbaren Knotens hat einige technische Vorteile gegenüber dem Stand der Technik. So werden z.B. beschriebene Aspekte der Sicherheit, der Performanz oder auch der Fehlertoleranz verbessert.
Durch die Nutzung von Standardsystemen als Prozessrechner – in Kombination mit einer externen Schnittstelle, die auf ebenfalls in der Standardisierung befindlichen Web-Services aufbaut – ergibt sich aber noch ein weiterer betriebswirtschaftlicher Vorteil einer derartigen Architektur [BTM+04]. Die beschriebene Architektur eines programmierbaren Knotens ähnelt in Teilen den Vorschlägen, die Grid-Computing als Dienst über Web-Services anbieten (siehe z.B. [GAK+03]). Vergleicht man jene Vorschläge mit der in diesem Abschnitt präsentierten Architektur eines programmierbaren Knotens, findet man folgende Gemeinsamkeiten:

- Externe Schnittstelle über standardisierte Web-Services.
- Fremder Code wird geladen und ausgeführt.

Während Programmierbare Netze zum Ziel haben, Dienste durch das Netz anzubieten, die aktiv in Datenströme eingreifen, um z.B. (wie in Kapitel 5 beschrieben) die Fehlertoleranz bei der Kommunikation zu verbessern, haben Grid-Computing-Systeme grundsätzlich ein anderes Ziel: Durch Grid-Computing soll es ermöglicht werden, einen – in der Regel umfangreichen – Rechenprozess durch über ein Netz verfügbare Ressourcen berechnen zu lassen. Das Ergebnis dieses Rechenprozesses wird dann von entsprechenden Ressourcen zurückgeliefert. Dies kann z.B. die numerische Lösung einer komplexen Gleichung sein.
Der dafür notwendige Datenaustausch kann mittels Web-Services durchgeführt werden: Auf die Grid-Ressourcen wird dadurch Code geladen, der den Rechenprozess darstellt, und ausgeführt. Dieses Prinzip ähnelt dem der Programmierbaren Netze: Auch in diesem Fall wird Code (Dienste) geladen und ausgeführt.

Die Entwicklung des Grid-Computing steht gegenwärtig vor zwei Problemen [FH03]. Zum einen gibt es noch offene Problemstellungen bei der Entwicklung einer geeigneten Middleware. Insbesondere die Sicherheit und Zuverlässigkeit stellt hier Probleme dar. Zum anderen ist für eine Vielzahl von Anwendungen für Grid-Computing eine hochbitratige Datenverbindung wünschenswert, da u.U. sehr große Datenmengen übertragen werden müssen.

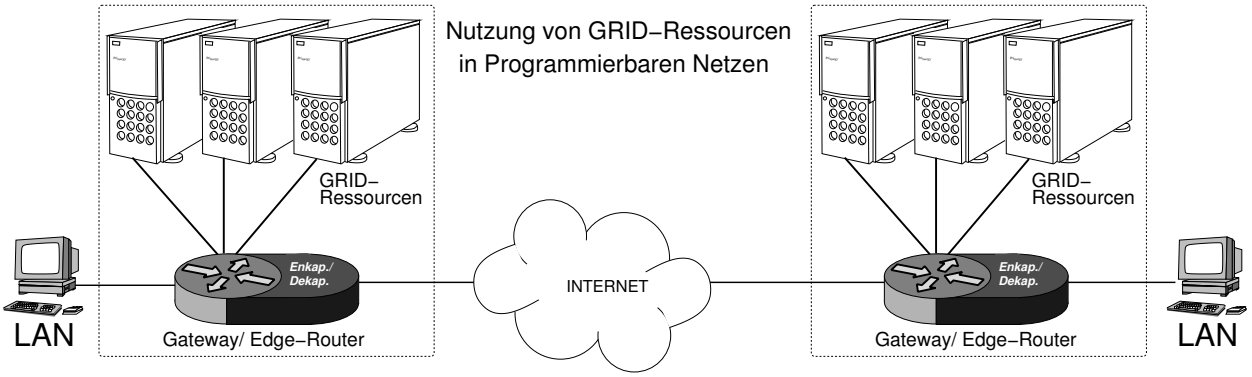

Abbildung 6.6: Positionierung von Grid-Ressourcen

Dem Wunsch der möglichst schnellen Anbindung von Grid-Ressourcen, könnte man dadurch begegnen, dass man (wie in Abbildung 6.6 beschrieben) die Grid-Ressourcen direkt neben Access-Routern bzw. Gateways im Netz platziert. Die Grid-Ressourcen wären dann an der gleichen Stelle platziert wie die in Abbildung 6.4 dargestellten Prozessrechner der Rechner-Komponente des programmierbaren Knotens. Somit hätte man nicht nur auf abstrakter Ebene eine Ähnlichkeit, sondern auch in der tatsächlichen Bereitstellung der Systeme.

Mit der in diesem Kapitel beschriebenen, komponentenbasierten Architektur eines programmierbaren Knotens wäre es ohne größere Adaption realisierbar, auch Grid-Computing-Dienste anzubieten. Die Router-Komponente müsste für diesen Fall nicht konfiguriert werden, da eine Enkapsulierung von Datenpaketen unnötig ist.
Aus einem anderen Blickwinkel betrachtet, wäre es möglich, bestehende Grid-Computing-Systeme zu einem komponentenbasierten, programmierbaren Knoten zu erweitern. Lediglich die Router-Komponente müsste entsprechend erweitert werden.

Der wesentliche Vorteil liegt somit in einer möglichen gemeinsamen Nutzung von Ressourcen, sowohl für Grid-Computing als auch für die Bereitstellung von Programmierbaren Netzen. Wir sehen in dieser Möglichkeit einen großen Vorteil der Architektur gegenüber dem Stand der Technik, da eine Verbreitung der Technologie der Programmierbaren Netze damit kostengünstiger zu bewerkstelligen wäre.

6.3 Fehlertoleranzdienst und Plattform

In diesem Abschnitt gehen wir auf das Zusammenspiel des in Kapitel 5 beschriebenen Fehlertoleranzdienstes mit der programmierbaren Plattform ein.
Zunächst geben wir einen Überblick über die gesamte Softwarearchitektur in einem Zugangsknoten, bestehend aus dem statischen Teil der programmierbaren Plattform und dem dynamisch ladbaren Dienst.
Der dynamisch ladbare Fehlertoleranzdienst unterteilt sich selbst in ein statisches Framework und adaptive flow-spezifische Module.
Des Weiteren gehen wir noch kurz auf die Verbesserung von Sicherheitsaspekten und der Fehlertoleranz bei der Bereitstellung des Dienstes ein.

6.3.1 Softwarearchitektur in einem Zugangsknoten

In Abbildung 6.7 ist der in Abschnitt 5.1.1 beschriebene Zugangsknoten zum Fehlertoleranzdienst dargestellt als geladener Dienst auf der komponentenbasierten, programmierbaren Plattform. Aus Gründen der besseren Übersicht ist das Framework des Dienstes und nur ein flow-spezifisches Modul dargestellt. Da für jede durch den Fehlertoleranzdienst geschützte

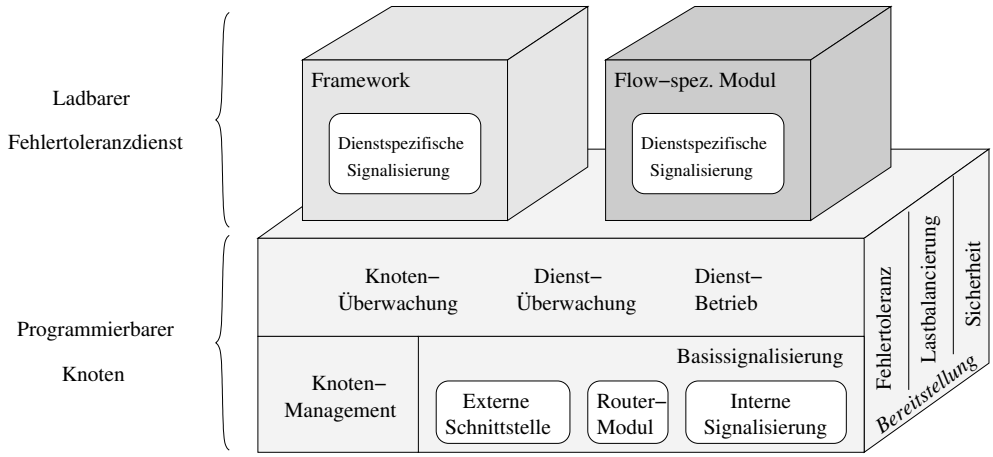

Abbildung 6.7: Zusammenspiel von Dienst und Plattform

Verbindung ein eigens flow-spezifisches Modul gestartet werden muss, ist in einem realen Betrieb mit einer Vielzahl von derartigen Modulen zu rechnen, die parallel ablaufen.
Der Ladevorgang bzw. der Ablauf des Fehlertoleranzdienstes gestaltet sich folgendermaßen:

- Das Framework des Fehlertoleranzdienstes wird geladen, konfiguriert und ausgeführt.
- Im Anschluss startet die Analyse des Datenverkehrs. Parallel dazu werden die Verbindungen zu den entsprechenden Overlay-Netzen aufgebaut.
- Soll für eine bestimmte Instanz einer Applikation der Fehlertoleranzdienst aktiv werden, dann initiiert das Framework den Start eines flow-spezifischen Moduls, das diese Aufgabe übernimmt.

Das Framework stellt die Basis des Fehlertoleranzdienstes dar und wird über die in Abschnitt 6.2.1 beschriebene, externe Schnittstelle des programmierbaren Knotens geladen. Im Anschluss verbindet sich das Framework mit den in Kapitel 5 beschriebenen Overlay-Netzen. Ist die Verbindung zu den entsprechenden Overlay-Netzen hergestellt, dann ist der Fehlertoleranzdienst einsetzbar.

Im Framework müssen bestimmte Vorgehensweisen (engl. policies) entsprechend konfiguriert werden. Dies kann z.B. sein, bestimmte HTTP-basierte Datenverbindungen aus dem konnektierten LAN mit dem Internet zu schützen. Die Möglichkeiten der Konfiguration sind letztlich abhängig vom Betreibermodell des Fehlertoleranzdienstes. Folgende Fragen sind in diesem Zusammenhang zu klären:

- Ist es gewünscht, dass einzelne Benutzer den Dienst speziell für ihre Anforderungen konfigurieren dürfen? Dies kann z.B. eine Kombination von horizontaler und vertikaler Fehlermaskierung für eine bestimmte Anwendung sein, mit dem Wunsch, ein spezielles Software-Modul für die vertikale Fehlermaskierung bzw. Fehlerabschwächung zu verwenden.
- Soll die Konfiguration ausschließlich von einem Administrator vorgenommen werden?

Auf derartige Problemstellungen des täglichen Betriebs gehen wir in dieser Arbeit nicht ein. Die vorgeschlagene Architektur ist jedoch flexibel gehalten, um diese Fälle abdecken zu können.

Abhängig von der jeweiligen Konfiguration werden alle entsprechenden Datenpakete, die zwischen konnektiertem LAN und dem Internet ausgetauscht werden, dem Framework zur Analyse übergeben. Das Framework verhält sich somit ähnlich wie eine Firewall.
Im Folgenden beschreiben wir exemplarisch den Start eines flow-spezifischen Moduls des Fehlertoleranzdienstes das, gemäß Konfiguration, eine HTTP-Verbindung schützt:

- Der Analysator des Frameworks erkennt den, dem HTTP vorausgehenden, Aufbau einer TCP-Verbindung mit dem Internet. Laut Konfiguration ist die HTTP-Verbindung z.B. durch eine horizontale Fehlermaskierung zu schützen.
- Das Framework initiiert daraufhin den Start eines flow-spezifischen Moduls, das für diese eine Verbindung zuständig sein wird.
- Das in Abschnitt 5.3.8 beschriebene dynamische RON wird konfiguriert.
- Das flow-spezifische Modul wird Teil des dynamischen RON.
- Zum Abschluss initiiert das Framework die direkte Weiterleitung der betreffenden Daten an das gestartete Modul.

Das Framework startet das Modul nur indirekt. Mittels der Basissignalisierung teilt es den Steuerungs- bzw. Signalisierungsrechnern mit, dass ein Modul gestartet werden soll. Diese starten auf einem der Prozessrechner dann das gewünschte flow-spezifische Modul als eigenständiges Userspace-Programm, wie in Abschnitt 6.1 beschrieben. Die Steuerungsrechner können eine Lastbalancierung vornehmen, da das flow-spezifische Modul auf einem beliebigen Prozessrechner ausgeführt werden kann.
Im Anschluss konfiguriert der Steuerungsrechner noch die Router-Komponente und lenkt damit den entsprechenden Datenstrom nun direkt zum gestarteten Modul am Framework vorbei. Das flow-spezifische Modul ist dann allein für die Bereitstellung der gewünschten Fehlermaskierung verantwortlich.
Sowohl das flow-spezifische Modul als auch das Framework können über die dienstspezifische Signalisierung Informationen austauschen. Sobald die dynamischen Overlay-Netze entsprechend konfiguriert sind, beginnt die gewünschte Fehlermaskierung.

Das flow-spezifische Modul überwacht eigenständig sowohl die zu schützende Datenverbindung als auch das für diese Verbindung konfigurierte, dynamische Overlay-Netz. Tritt eine gravierende Veränderung auf, z.B. ein Ausfall innerhalb des Overlay-Netzes, meldet das flow-spezifische Modul dies dem Framework des Fehlertoleranzdienstes. Dort wird entsprechend reagiert, es kann z.B. ein weiteres dynamisches RON konfiguriert werden.

6.3.2 Fehlertoleranz und Sicherheit durch Knotenarchitektur

In diesem Abschnitt gehen wir auf Aspekte der Fehlertoleranz und Sicherheit der Knotenarchitektur ein. Die grundlegende Philosophie der Architektur ist es, eine zuverlässige programmierbare Plattform zur Verfügung zu stellen. Auf dieser Plattform kann dann der in Kapitel 5 beschriebene Dienst angeboten werden, der Fehlertoleranz gegenüber den Applikationen bzw. gegenüber Applikationsfehlern bereitstellt.

Die Zuverlässigkeit bzw. Fehlertoleranz des programmierbaren Knotens betrachten wir unter zwei Gesichtspunkten:
Zum einen gehen wir auf Versagen ein, das auf Ausfälle innerhalb des programmierbaren Knotens zurückgeht die nicht durch Dienste verursacht sind. Dies kann z.B. ein (Hardware-) Ausfall in einem Prozessrechner sein.
Zum anderen betrachten wir das Versagen des Systems, das auf Fehlverhalten von geladenen Diensten zurückzuführen ist. Dies kann z.B. ein absichtlicher oder unabsichtlicher Angriff eines Dienstes auf den programmierbaren Knoten sein.
Angriffe, die ihren Ursprung außerhalb des programmierbaren Knotens haben, berücksichtigen wir in dieser Arbeit nicht. Die in Abschnitt 2.4.1 beschriebenen Ansätze gehen auf Problemstellungen dieser Art ein.

Die koordinierende Aufgabe innerhalb des komponentenbasierten, programmierbaren Knotens übernehmen die Steuerungs- bzw. Signalisierungsrechner. Auf diesen Rechnern läuft ausschließlich proprietäre Software des programmierbaren Knotens. Dienste dürfen dort nicht geladen und ausgeführt werden. Diese Rechner sind redundant ausgelegt. Fällt einer der Steuerungsrechner aus, dann übernimmt ein anderer die Aufgabe. Die Steuerungsrechner stellen die Schnittstelle des programmierbaren Knotens nach außen dar. Mittels der in Abschnitt 3.5.1 vorgestellten Verfahren können Ausfälle der externen Schnittstelle maskiert werden. Intern wird zwischen den Steuerungs- bzw. Signalisierungsrechnern durch die Basissignalisierung entsprechende Information ausgetauscht. Somit kann im Falle des Ausfalls des Master-Steuerungsrechners ein anderer die Funktion übernehmen, ohne das eine nennenswerte Unterbrechung des Betriebs erfolgt. Somit ist die Steuerung des programmierbaren Knotens fehlertolerant ausgelegt.

Dienste werden ausschließlich auf Prozessrechnern in der Rechner-Komponente ausgeführt. Auch hier ist Fehlertoleranz durch die Architektur des Knotens gegeben. Kommt es zu einem Ausfall eines der Prozessrechner, dann müssen die auf dieser Maschine laufenden Dienste von einem anderen Prozessrechner zur Verfügung gestellt werden.
Prinzipiell können in diesem Fall, wie in Abschnitt 3.2 beschrieben, Ersatzschaltverfahren bei der Bereitstellung des Dienstes angewandt werden.
Das 1+1 Ersatzschalten bzw. *Hot-Standby* für Dienste startet parallel zu einem Master-Dienst einen identischen Ersatz-Dienst auf einem anderen Prozessrechner. Nur der Master-Dienst bearbeitet den entsprechenden Datenstrom. Über die Basissignalisierung und auch die dienstspezifische Signalisierung wird der Ersatz-Dienst auf dem gegenwärtigen Stand gehalten. Fällt nun der Master-Dienst bzw. der zugrunde liegende Prozessrechner aus, dann kann die Steuerung des programmierbaren Knotens schnell ersatzschalten, indem die Router-Komponente entsprechend umkonfiguriert wird. Der zu bearbeitende Datenstrom wird dann zum Ersatz-Dienst umgeleitet. Für den Fall des 1:1 Ersatzschaltens muss der entsprechende Ersatz-Dienst erst auf einem verfügbaren Prozessrechner initiiert werden.

Haben die betreffenden Dienste atomaren Charakter, ist es seitens der Steuerung des programmierbaren Knotens lediglich notwendig, die Router-Komponente neu zu konfigurieren, und die entsprechenden Dienste auf einem anderen, funktionsfähigen Prozessrechner zur Verfügung zu stellen.
Für den Fall eines komplexen, nicht-atomaren Dienstes (wie z.B. dem in dieser Arbeit beschriebenen Fehlertoleranzdienst) gestaltet sich die fehlertolerante Bereitstellung komplexer. Verfügt ein Dienst über eine dienstspezifische Signalisierung, wie z.B. die Verbindung zu einem Overlay-Netz, muss dies bei einem Ausfall des zugrunde liegenden Prozessrechners berücksichtigt werden.
Wird durch das entsprechende Ersatzschaltverfahren von einem Master-Dienst zu einem Ersatz-Dienst auf einem verfügbaren Prozessrechner umgeschaltet, müssen die notwendigen Informationen vorliegen, um den Ersatz-Dienst entsprechend initialisieren zu können. Diese Informationen zur Initialisierung müssen vor dem Ausfall des Master-Dienstes durch die Basissignalisierung an die Steuerungs- und Signalisierungsrechner übermittelt werden.
Liegen diese Informationen nicht vor, dann wird der auf einen anderen Prozessrechner verlagerte Dienst von der Steuerung so behandelt, als ob der Dienst regulär gestartet werden würde.

Die Sicherheit der Knotenarchitektur gegenüber internen Angriffen von Diensten wird durch mehrere Punkte verbessert. Zum einen umgeht der in Abbildung 6.2 beschriebene Mechanismus der Enkapsulierung bzw. Dekapsulierung, die in Abschnitt 3.7.1 angeführte Interaktion zwischen dem Kern des Betriebssystems und dem Userspace. Dadurch kann ausschließlich auf ausgereifte Standardfunktionen des Betriebssystems, wie z.B. die Benutzung von UDP-Sockets, zurückgegriffen werden.
Zum anderen ist es ein großer Vorteil im praktischen Betrieb der komponentenbasierten Architektur, dass die Maschinen der Rechner-Komponente keine spezifischen Erweiterungen innerhalb des Betriebssystems benötigen. Jegliche Software (Dienste als auch Steuerung bzw. Signalisierung) läuft innerhalb der Rechner-Komponente im Userspace ab. Dadurch sind diese Maschinen unabhängig von Sicherheitsaspekten der jeweiligen Betriebssysteme. Das bedeutet konkret: Wird im Betriebssystem eine Sicherheitslücke bekannt, die z.B. die Erlangung von *Root-Rechten* ermöglicht, dann kann auf den Maschinen der Rechner-Komponente sofort ein entsprechendes Sicherheitsupdate durchgeführt werden, ohne den Betrieb des Knotens lange zu unterbrechen. Dieses Sicherheitsupdate verändert in der Regel den Kern des Betriebssystems und behebt dort die entsprechenden Fehler.
Praktische Erfahrungen während des Prototypenbaus mit den in Abschnitt 3.7 präsentierten programmierbaren Knotenarchitekturen zeigten in diesem Punkt ein erhebliches Defizit. Dadurch, dass in diesen Architekturen Erweiterungen im Kern des Betriebssystems vorgenommen worden sind, um die programmierbare Funktionalität überhaupt zu ermöglichen, kann ein Standard-Sicherheitsupdate in diese speziell veränderten Kerne meist nicht eingebracht werden. Gleiches gilt auch für das Update des gesamten Kerns. Steht eine neue Version eines Betriebssystemkerns zur Verfügung, kann diese nicht ohne weiteres in die programmierbaren Knoten eingebracht werden. In der Regel sind in diesem Zusammenhang erst umfangreiche Änderungen der proprietären Kernerweiterungen notwendig, um die programmierbare Funktionalität wieder herzustellen.
Praktisch bedeutet dies in der Regel, dass derartige Architekturen meist auf veralteten – und damit u.U. anfälligen – Kernen eines Betriebssystems laufen. Durch unsere Architektur um-

gehen wir dieses Problem, da die Maschinen nicht von der Version des Betriebssystemkerns abhängig sind.

Ein weiterer Vorteil, gegenüber den in Abschnitt 3.7 präsentierten Architekturen, ist die Aufteilung von Steuerung und Bereitstellung der Dienste auf unterschiedlichen Maschinen innerhalb der Rechner-Komponente. Gegenüber der Knotenarchitektur feindlich gesinnte Dienste haben dadurch keine Möglichkeit direkt auf die Steuerung des programmierbaren Knotens einzuwirken. Ein Dienst kann lediglich das Verhalten des Prozessrechners beeinflussen, auf dem er abläuft. Störungen der Steuerung durch Dienste, z.B. durch Blockierung bestimmter Ressourcen, sind dadurch eingeschränkt.

Dadurch, dass unsere Architektur keine zusätzliche Softwareschicht als *Execution-Environment (EE)* einführt (wie dies in userspace-basierten programmierbaren Knoten üblich ist) ergibt sich ein weiterer Vorteil.
Jeder Dienst wird auf den Prozessrechnern unter einem eineindeutigen Benutzernamen gestartet. Da der Dienst nur einen sehr begrenzten Umfang an Rechten benötigt, ist es möglich, die Rechte des Dienstes somit durch entsprechend limitierte Benutzerrechte einzuschränken. Somit werden die Rechte des Dienstes durch Standardfunktionen des Betriebssystems begrenzt.
In Architekturen, die ein dediziertes EE einführen, ist dies nicht möglich. Ein EE benötigt umfangreiche Rechte; die gestarteten Dienste erben diese Rechte. Durch den Einbau spezieller Sicherheitsfunktionen innerhalb des EE, werden die Rechte der Dienste dann begrenzt. Derartige Architekturen bauen somit nicht auf Standardsicherheitsfunktionen des Betriebssystems auf, sondern implementieren diese selbst. Werden neue Bedrohungsszenarien bekannt, dann sind in der Regel relativ schnell Updates erhältlich, die Standardfunktionen entsprechend anpassen. Greift man aber nicht auf diese Standardfunktionen zurück, ist es erforderlich die notwendige Behandlung von neuen Bedrohungsszenarien im EE selbst zu implementieren.

6.4 Zusammenfassung

Dieses Kapitel stellt eine neue, komponentenbasierte Architektur eines programmierbaren Knotens vor. Durch eine relativ einfach zu realisierende Erweiterung schlagen wir vor, einen Standard-Router zur Router-Komponente umzufunktionieren. Dieser Router-Komponente stellen wir eine Rechner-Komponente bei, die aus einer Vielzahl von Standardrechnern bestehen kann. Die beiden Komponenten bilden zusammen eine programmierbare Knotenarchitektur. Die wesentlichen Beiträge dieses Kapitels sind:

- Konzept der Aufteilung von Routing, Steuerung und Bereitstellung der Dienste auf unterschiedlichen Maschinen.
- Enkapsulierungs-/ Dekapsulierungs-Modul zur Erweiterung eines Standard-Routers.
- Fehlertolerante Steuerungs- bzw. Signalisierungsmechanismen.
- Fehlertolerante Bereitstellung von atomaren als auch komplexen Diensten mit dienstspezifischer Signalisierung.
- Verbesserung der Sicherheit eines programmierbaren Knotens gegenüber böswilligen Diensten.
- Durch die lokale Skalierbarkeit der Architektur wird die Leistungsfähigkeit des programmierbaren Knotens gesteigert. Es können parallel eine Vielzahl an Diensten bereitgestellt werden.
- Ausschließliche Benutzung von Standardbetriebssystemen bzw. deren Funktionen, bei der Bereitstellung von Diensten. Insbesondere ist die Architektur unabhängig von den Versionen der verwendeten Betriebssysteme.
- Durch die parallele Nutzung von Ressourcen für Programmierbare Netze und Anwendungen des Grid-Computing ergeben sich Kostenvorteile.

Das in der Funktionalität, und damit in der Komplexität, bewusst begrenzt gehaltene Enkapsulierungs-/ Dekapsulierungs-Modul lässt sich in einen Standard-Router integrieren. Diese, im Vergleich zu den in Abschnitt 3.7 präsentierten Architekturen, relativ einfache Erweiterung eines Standard-Routers sehen wir als Hauptbeitrag dieses Kapitels.
Durch die beschriebene Architektur ist es nicht notwendig auf einem Router ein komplexes, dediziertes Betriebssystem zu installieren, das es erlaubt, Dienste zu laden und auszuführen. Der Router bzw. die Router-Komponente konzentriert sich somit im Wesentlichen auf das Routing.
Darüber hinaus erlaubt die komponentenbasierte Architektur die Verbesserung der Sicherheit und Fehlertoleranz beim Betrieb des programmierbaren Knotens. Durch die ausschließliche Benutzung von Standardbetriebssystemen in der Rechner-Komponente sehen wir des Weiteren mögliche Kosteneinsparungen beim Betrieb, da keine proprietäre Software im Kern des Betriebssystems angepasst werden muss.

Kapitel 7

Zusammenfassung

Der Siegeszug der Informationstechnologien der letzten Jahre brachte im wirtschaftlichen und im privaten Bereich große Veränderungen mit sich.
So werden Unternehmen immer stärker miteinander vernetzt. Mittlerweile ist es Standard, dass man z.B. bei der Bestellung eines Produktes über das Internet, indirekt mit den Unterlieferanten des Verkäufers verbunden ist. Dadurch kann eine zeitlich optimierte Produktion bzw. Lieferung des Produktes gewährleistet werden, bei möglichst niedriger Lagerhaltung des Verkäufers bzw. Systemlieferanten. Dies gilt sowohl für Geschäftsbeziehungen im Business-to-Business (B2B) als auch im Business-to-Consumer (B2C).
Auch im privaten Bereich spielt die Vernetzung eine immer größere Rolle. Als Beispiel sei hier ein bekannter Spruch aus der Werbung eines Telekommunikationsunternehmens angeführt: *"Ich leb' online"*. Ein immer größer werdender Prozentsatz der Bevölkerung hat Zugang zum Internet, und benutzt dieses auch intensiv mit unterschiedlichsten Anwendungen.

Die steigende Verbreitung und Akzeptanz des Internets bringt aber auch Gefahren mit sich. So werden wir in immer stärkerem Maß abhängig von der Verfügbarkeit des Mediums Internet. Zukünftige Anwendungen könnten diesen Effekt noch verstärken.

Fehlertoleranz wurde bei der Entwicklung des Internets von Beginn an mit einbezogen. So gibt es domänenübergreifende Fehlertoleranz, die auf gegebener Pfaddisjunktivität aufbaut. Mittels Restaurationsverfahren können ausgefallene Regionen im Netz damit umgangen werden.
Die dafür notwendige Konvergenzzeit kann von vielen verteilten Applikationen nicht toleriert werden. Aus diesem Grund wurden in jüngster Zeit Overlay-Netze vorgeschlagen, die ein Ersatzschalten ermöglichen, und somit den Restaurationsverfahren zeitlich überlegen sind. Die beschriebenen Ansätze beziehen aber Applikationen nicht direkt ein. Sie sind darüber hinaus in der Skalierbarkeit begrenzt.

An diesem Punkt setzt diese Arbeit an. Wir schlagen eine adaptive, mehrdimensionale Fehlertoleranz als Netzdienst vor. Durch kooperierende Overlay-Netze wird die Signalisierung, bei der Suche nach alternativen Pfaden durch das Internet, von der Bereitstellung der Fehlermaskierung entkoppelt. Dadurch wird Skalierbarkeit erreicht. Die Fehlermaskierung durch Ersatzschalten erfolgt in dynamischen, flow-basierten, minimalen Overlay-Netzen.

Des Weiteren erlaubt der flow-basierte Ansatz die Integration von contentspezifischer Fehlermaskierung bzw. -abschwächung. Durch diesen Crosslayer-Ansatz können Applikationsfehler spezifisch maskiert werden. Wesentlicher Punkt ist die Einbindung von im Netz vorhandenen Mechanismen, die Repliken zur Verfügung stellen, wie z.B. CDN bzw. Cacheing-Systeme.
Zusätzlich können Applikationen direkt mit dem Fehlertoleranzdienst interagieren. Dadurch ist eine detaillierte Anpassung an Bedürfnisse der Applikationen möglich.

Der beschriebene Fehlertoleranzdienst kann im Netz vollkommen transparent gegenüber Endsystemen bereitgestellt werden. Wir sehen hier eine wesentliche Voraussetzung für eine mögliche allmähliche Migration des Dienstes in das Internet, da Endsysteme bzw. Applikationen nicht an den Fehlertoleranzdienst angepasst werden müssen.
Seitens des Fehlertoleranzdienstes ist aber eine Anpassung an die zu schützenden Applikationen notwendig. Aus diesem Grund schlagen wir den Fehlertoleranzdienst als Dienst in Programmierbaren Netzen vor, die Adaptivität gewährleisten. Eine möglichst rasche Adaption des Fehlertoleranzdienstes ist in zwei Punkten notwendig. Zum einen müssen dem Dienst neue Protokolle und Anwendungen bekannt sein, um entsprechende Applikationen behandeln zu können. Zum anderen kann es auch erforderlich sein die beschriebenen dienstinternen Algorithmen an die tatsächlichen Gegebenheiten, im Laufe des Betriebs des Fehlertoleranzdienstes, wie z.B. die Verteilung und Nutzung, anzupassen. Damit ist eine Bereitstellung des Dienstes im Sinne einer *Rapid-Service-Creation* möglich.

Grundlegend für eine Bereitstellung des Dienstes ist die, in dieser Arbeit eingeführte, skalierbare und fehlertolerante programmierbare Knotenarchitektur. Der beschriebene Ansatz bildet aus erweiterten Standard-Routern, in Kooperation mit Standard-Rechenmaschinen, eine komponentenbasierte, programmierbare Knotenarchitektur.
Der Ansatz erlaubt lokale Skalierbarkeit und damit die Bereitstellung von rechenintensiven, komplexen Diensten. Durch eine gekapselte und redundant ausgelegte Steuerung des Knotens ist eine fehlertolerante Bereitstellung von Diensten möglich.
Die komponentenbasierte Architektur ermöglicht des Weiteren die Bereitstellung der Dienste unabhängig, von der verwendeten Version der jeweiligen Betriebssysteme, zu organisieren. Dies verbessert die Sicherheit der programmierbaren Knotenarchitektur.
Diese grundlegenden Eigenschaften sind notwendig, um den beschriebenen Fehlertoleranzdienst auf solider Basis im Netz zur Verfügung zu stellen.

In der vorgeschlagenen Architektur eines programmierbaren Knotens sehen wir noch einen weiteren Vorteil. Durch die Abstrahierung der Netzebenen, bezüglich der Bereitstellung von Diensten, ist es möglich, Grid-Computing-Ressourcen in den programmierbaren Knoten zu integrieren. Von einer anderen Sichtweise betrachtet, können vorhandene Grid-Computing-Architekturen relativ einfach zu programmierbaren Knoten erweitert werden.
In dieser möglichen, einfachen Konvergenz der Technologien der Programmierbaren Netze und des Grid-Computing sehen wir potentielle Synergieeffekte.
Die Nachfrage nach Grid-Computing-Systemen wird bereits durch erste, kommerziell erhältliche Produkte befriedigt. Verschiedene Studien sagen diesem Markt in den nächsten Jahren ein großes Wachstum voraus.
Betrachtet man hingegen die Technologie der Programmierbaren Netze, so findet man bisher keine Anwendungen außerhalb von Forschungsnetzen. Den Grund dafür sehen wir, unter anderem, in einem Kausalitätsproblem: Die Bereitstellung von komplexen, netzübergreifenden

Diensten in Programmierbaren Netzen macht im Wesentlichen nur Sinn, wenn programmierbare Plattformen quasi überall zur Verfügung stehen. Das grundlegende Problem liegt darin, dass programmierbare Plattformen in großer Zahl parallel eingeführt werden müssten, um entsprechende Dienste anbieten zu können. Somit müssten bevor erste Dienste angeboten werden können (und somit mögliche Einnahmen generiert werden können), relativ große Investitionen getätigt werden. Dieses Investitionsrisiko erweist sich in der Praxis anscheinend als große Hürde.
Durch eine Konvergenz mit bestehenden Architekturen von Grid-Systemen sehen wir hier eine risikobegrenzte Möglichkeit, um die Technologie der Programmierbaren Netze (überhaupt) im Netz installieren zu können.

Sowohl der beschriebene Fehlertoleranzdienst als auch die komponentenbasierte, programmierbare Knotenarchitektur wurden prototypisch implementiert. Dadurch wurde die Machbarkeit der vorgestellten Konzepte und Architekturen praktisch evaluiert.

Literaturverzeichnis

[AAH+98] Alexander, D. Scott, William A. Arbaugh, Michael W. Hicks, Pankaj Kakkar, Angelos D. Keromytis, Jonathan T. Moore, Carl A. Gunter, Scott M. Nettles, und Jonathan M. Smith: *The SwitchWare active network architecture*. In: *IEEE Network Special Issue on Active and Controllable Networks*, August 1998. 18, 50

[ABE+94] Albanese, Andres, Johannes Blömer, Jeff Edmonds, Michael Luby und Madhu Sudan: *Priority encoding transmission*. In: *35th IEEE Annual Symposium on Foundations of Computer Science*, Santa Fe, NM, USA, November 1994. 46

[ABEK+01] Alvisi, Lorenzo, Thomas Bressoud, Amr El-Khasab, Keith Marzullo und Dmitrii Zagorodnov: *Wrapping Server-Side TCP to Mask Connection Failures*. In: *INFOCOMM 2001*, Anchorage, Alaska, April 2001. 44

[ABKM01] Andersen, David G., Hari Balakrishnan, M. Frans Kaashoek und Robert Morris: *Resilient Overlay Networks*. ACM Symposium on Operating Systems Principles, Oktober 2001. 37, 42, 93, 99, 151

[AC98] Almeida, J. und P. Cao: *Measuring Proxy Performance with the Wisconsin Proxy Benchmark*. Journal of Computer Networks and ISDN Systems, 30(22), November 1998. 106

[AK00] Autenrieth, Achim und Andreas Kirstädter: *Fault-Tolerance and Resilience Issues in IP-Based Networks*. In: *Second International Workshop on the Design of Reliable Communication Networks, DRCN 2000*, Munich, Germany, April 2000. 6

[aka04] *Akamai Technologies Inc. Edgesuite*, Januar 2004. `http://www.akamai.com/en/html/services/edgesuite.html`. 39

[AMK+01] Alexander, D. Scott, Paul B. Menage, Angelos D. Keromytis, William A. Arbaugh, Kostas G. Anagnostakis und Jonathan M. Smith: *The Price of Safety in an Active Network*. Journal of Communications and Networks (JCN), special issue on programmable switches and routers, 2001. 52

[Aut03] Autenrieth, Achim: *Differntiated Resilience in IP-based Multilayer Transport Networks.* In: *Dissertation, Technische Universität München*, April 2003. 33

[Bac03] Bachmeir, Christian: *Fault-Tolerant 2-tier P2P based service discovery in PIRST-ON.* In: *10th IEEE International Conference on Telecommunications, ICT'2003*, Papeete, French Polynesia, Februar 2003. 88, 145, 151

[BACS02] Bagnulo, Marcelo, Bernardo Alarcos, Maria Calderon und Marifeli Sedano: *ROSA: Realistic Open Security Architecture for Active Networks.* In: *4th Annual International Working Conference on Active Networks, IWAN'2002*, Zurich, Switzerland, Dezember 2002. 53

[BBB02] Baumgartner, Florian, Torsten Braun und Bharat Bhargava: *Design and implementation of a Python-Based Active Network Platform for Network Management and Control.* In: *4th Annual International Working Conference on Active Networks, IWAN'2002*, Zurich, Switzerland, Dezember 2002. 51

[BBCW01] Branigan, S., H. Burch, B. Cheswick und F. Wojcik: *What Can You Do with Traceroute?* In: *IEEE Internet Computing*, September 2001. 90, 149

[BLM02] Byers, J., M. Luby und M. Mitzenmacher: *A Digital Fountain Approach to the Reliable Distribution of Bulk Data.* IEEE Journal on Selected Areas in Communications, Oktober 2002. 45

[BLP+02] Backx, Peter, Thijs Lambrecht, Liesbeth Paters, Bart Dhoedt und Piet Demeester: *Adaptive Distributed Caching.* In: *Short Paper Session in IEEE OPENARCH'2002*, New York, NY, USA, Juni 2002. 23

[BLSZ03] Bless, R., G. Lichwald, M. Schmidt und M. Zitterbart: *Fast Scoped Rerouting for BGP.* In: *IEEE International Conference on Networks*, Sydney, Australia, Oktober 2003. 35

[BM02] Broder, A. und M. Mitzenmacher: *Network Applications of Bloom Filters: A Survey.* In: *40th Annual Allerton conference on Communication, Control, and Computing*, Allerton Park, Illinois, USA, Oktober 2002. 105

[BPV+03] Bachmeir, Christian, Jianxiang Peng, Hans-Jörg Vögel, Chris Wallace und Gavin Conran: *Diversity Protected, Cache based Reliable Content Distribution - building on scalable, P2P and Multicast based Content Discovery.* In: *IEEE International Conference on High Speed Networks and Multimedia Communications, HSNMC 2003*, Estoril, Portugal, Juli 2003. 104, 105, 145

[Bre01] Breitling, Max: *Formale Fehlermodellierung für verteilte reaktive Systeme.* In: *Dissertation, Technische Universität München*, Juni 2001. 25, 26

[BT02] Bachmeir, Christian und Peter Tabery: *PIRST-ONs: A Service Architecture for Embedding and Leveraging Active and Programmable Networks.* In: *International Conference on Software, Telecommunications and Computer*

Networks, SoftCOM 2002, Split, Dubrovnik (Croatia), Ancona, Venice (Italy), Oktober 2002. 145

[BT03] Bachmeir, Christian und Peter Tabery: *Scalable Diverse Protected Multicast as a Programmable Service Leveraging IP Multicast*. In: *2nd IEEE International Symposium on Network Computing and Applications, NCA'03*, Cambridge, MA, USA, April 2003. 99, 145

[BTK03] Bachmeir, Christian, Peter Tabery und Johannes Käfer: *Towards Diverse Protection of Data Streams in Programmable Application Layer Overlay Networks*. In: *International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2003*, Split, Dubrovnik (Croatia), Ancona, Venice (Italy), Oktober 2003. 109, 145

[BTM+04] Bachmeir, Christian, Peter Tabery, Dimitar Marinov, Georgi Nachev und Jörg Eberspächer: *A Convergence Architecture for GRID Computing and Programmable Networks*. In: *Int. Workshop on Active and Programmable Grids Architectures and Components, APGAC'04*, Krakow, Poland, Juni 2004. 115, 118, 123, 145

[BTS+03] Bachmeir, Christian, Peter Tabery, Elie Sfeir, Dimitar Marinov, Georgi Nachev, Stephan Eichler und Jörg Eberspächer: *HArPooN: A Scalable, High Performance, Fault Tolerant Programmable Router Architecture*. In: *Poster Session in IFIP-TC6 5th Annual International Working Conference on Active Networks, IWAN'2003*, Kyoto, Japan, Dezember 2003. 114, 145

[BTÜS03] Bachmeir, Christian, Peter Tabery, Serdar Üzümcü und Eckehard Steinbach: *A Scalable Virtual Programmable Real-Time Testbed for Rapid Multimedia Service Creation and Evaluation*. In: *IEEE 2003 International Conference on Multimedia and Expo, ICME'03*, Baltimore, MD, USA, Juli 2003. 119, 145

[cFLKP99] Feng, Wu chi, M. Liu, B. Krishnaswami und A. Prabhudev: *A Priority-Based Technique for the Best-Effort Delivery of Stored Video*. In: *SPIE/IS&T Multimedia Computing and Networking Conference*, San Jose, CA, USA, Januar 1999. 69

[Chr02] Christofferson, Michael: *Get High Availability Using Effective Fault Management*. Journal of Communication System Design, 2002. 2, 108

[CKK02] Chen, Yan, Randy H. Katz und John D. Kubiatowicz: *SCAN: A Dynamic, Scalable, and Efficient Content Distribution Network*. In: *1st International Conference on Pervasive Computing*, Zurich, Switzerland, August 2002. 41

[CMK+99] Campbell, Andrew T., Herman G. De Meer, Michael E. Kounavis, Kazuho Miki, John B. Vicente, und Daniel Villela: *A Survey of Programmable Networks*. ACM SIGCOMM Computer Communications Review, 29(2):7–23, April 1999. 18

[Com00] Comer, Douglas E.: *Internetworking with TCP/IP: Volume I, Principles, Protocols, and Architecture*. Prentice Hall, Fourth Auflage, 2000. 6, 8

[CSF+03] Conrad, M., M. Schöller, T. Fuhrmann, G. Bocksch, und M. Zitterbart: *Multiple Language family Support for Programmable Network Systems*. In: *Proceedings of the 5th Annual International Working Conference on Active Networks, IWAN'2003*, Kyoto, Japan, Dezember 2003. 162

[CW02] Cogdon, Stephen und Ian Wakeman: *How Bad Are Overlay Networks?* In: *The London Communication Symposium, LCS2002*, September 2002. 15

[Dam04] Damodar, Narkhede Dinesh: *A Case Study: Internet Host Reliability*. In: *Technical Report: Interdisciplinary Program in Reliability Engineering, Indian Institute of Technology*, Bombay, India, Januar 2004. 2, 108

[DH98] Deering, S. und R. Hinden: *RFC 2460: Internet Protocol, Version 6 (IPv6) Specification*, Dezember 1998. 12

[DPP99] Decasper, D., G. Parulkar und B. Plattner: *A Scalable, High Performance Active Network Node*. In: *IEEE Network*, Januar 1999. 51

[Ebe02] Eberspächer, Jörg: *Handbuch für die Telekommunikation*, Kapitel Systemtechnische Grundlagen, Seiten 1–38–1–168. Springer-Verlag Berlin Heidelberg New York, 2nd Auflage, 2002. 29, 32

[Eri94] Eriksson, Hans: *Mbone: the multicast backbone*. In: *Communications of the ACM, volume 37*, August 1994. 12, 13

[FCAB00] Fan, Li, Pei Cao, Jussara Almeida und Andrei Z. Broder: *Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol*. IEEE/ACM TRANSACTIONS ON NETWORKING, 8(3), Juni 2000. 106

[FH03] Frumkin, M. und R. Hood: *Using Grid Benchmarks for Dynamic Scheduling of Grid Applications*. In: *IASTED International Conference on Parallel and Distributed Computing and Systems, PDCS'03*, Marina del Rey, CA, USA, November 2003. 123

[FHSZ02] Fuhrmann, T., T. Harbaum, M. Schöller und M. Zitterbart: *AMnet 2.0: An Improved Architecture for Programmable Networks*. In: *4th Annual International Working Conference on Active Networks, IWAN'2002*, Zurich, Switzerland, Dezember 2002. 50, 87

[Fin03] Fink, Bob: *6Bone Overview and Links*, Juni 2003. `http://www.6bone.net`. 12

[FK99] Foster, Ian und Carl Kesselman: *Computational Grids*. In: *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufman, 1999. 15

[For95] Forum, ATM: *LAN Emulation over ATM 1.0, af-lane-0021.000*, January 1995. 13

[fS94] Standardization, International Organization for: *ISO/IEC 13818: Information Technology - Generic coding of moving pictures and associated audio (MPEG-2).* Geneva, Switzerland, 1994. 46

[Fuh03] Fuhrmann, Thomas: *Supporting Peer-to-Peer Computing with FlexiNet.* In: *Third International Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems*, Tokyo, Japan, Mai 2003. 16, 23

[GAK+03] Gannon, Dennis, Rachana Ananthakrishnan, Sriram Krishnan, Madhusudhan Govindaraju, Lavanya Ramakrishnan und Aleksander Slominski: *Grid Web Services and Application Factories, Chapter 9 in Grid Computing: Making the Global Infrastructure a Reality.* Wiley Europe Publishers, April 2003. 123

[Gla03] Glasmann, Josef: *Ressourcenmanagement für Echtzeitverkehre in Intranets.* In: *Dissertation, Technische Universität München*, November 2003. 61

[GLH+00] Gleeson, B., A. Lin, J. Heinanen, G. Armitage und A. Malis: *RFC 2764: A Framework for IP Based Virtual Private Networks*, Februar 2000. 13

[gnu] *Gnutella Protocol Specification v0.4., August 2000.* http://dss.clip2.com/GnutellaProtocol04.pdf. 23, 89

[Gör89] Görke, Winfried: *Fehlertolerante Rechensysteme.* In: *Handbuch der Informatik, Band 2.1, R. Oldenburg Verlag*, 1989. 28

[GSF95] Girod, B., E. Steinbach und N. Faerber: *Comparison of the H.263 and H.261 Video Compression Standards.* In: *Standards and Common Interfaces for Video Information Systems, K.R. Rao, editor, Critical reviews of optical science and technology, Volume CR60*, Philadelphia, Pennsylvania, USA, Oktober 1995. 51

[Har02] Harbaum, Till: *Rekonfigurierbare Routerhardware für adaptive Dienstplattformen.* In: *Dissertation, Shaker Verlag GmbH*, Achen, Mai 2002. 18, 51

[hCRSZ01] Chu, Yang hua, Sanjay G. Rao, Srinivasan Seshan und Hui Zhang: *Enabling Conferencing Applications on the Internet Using an Overlay Multicast Architecture.* In: *ACM SIGCOMM 2001*, San Diego, USA, August 2001. 14

[HE00] Hartman, John und Will Evans: *Better Fault Tolerance via Application-Enhanced Networks.* In: *DARPA Joint IA&S PI meeting July 19*, San Antonio, TX, USA, Juli 2000. 53

[Hei01] Heidtmann, Klaus: *Fault-Tolerant Real-Time Videocommunication via Best-Effort Networks.* In: *IST Network of Excellence in Distributed and Dependable Computing Systems, 4th Plenary Workshop*, Pisa, Italy, Oktober 2001. 46

[HJS03] Hess, A., M. Jung und G. Schäfer: *FIDRAN: A flexible Intrusion Detection and Response Framework for Active Networks.* In: *8th IEEE Symposium on Computers and Communications (ISCC'2003)*, Kemer, Antalya, Turkey, Juli 2003. 22, 61

[HK99] Hicks, Michael und Angelos D. Keromytis: *A Secure PLAN*. In: *First International Working Conference on Active Networks, (IWAN 1999)*, Berlin, Germany, Juni 1999. 52

[HSS+02] Hess, A., M. Schöller, G. Schäfer, M. Zitterbart und A. Wolisz: *A dynamic and flexible Access Control and Resource Monitoring Mechanism for Active Nodes*. In: *IEEE Openarch*, New York, NY, USA, Juni 2002. 52

[HSWZ01] Harbaum, Till, Anke Speer, Ralph Wittmann und Martina Zitterbart: *Providing Heterogeneous Multicast Services with AMnet*. Journal of Communications and Networks (JCN), special issue on programmable switches and routers, 2001. 23, 123

[Hus01] Huston, G.: *Analyzing the Internet's BGP Routing Table*. In: *The Internet Protocol Journal*, März 2001. 35

[ibm03] *IBM Grid Computing*, August 2003.
`http://www-1.ibm.com/grid/`. 15

[Ins81] Institute, Information Sciences: *RFC 793: TRANSMISSION CONTROL PROTOCOL*, September 1981. 44

[isc03] *The Internet Software Consortium (ISC), Internet Domain Survey*, Januar 2003.
`http://www.isc.org/ds/`. 7

[Ise99] Iselt, Andreas: *Ausfallsicherheit und unterbrechungsfreies Ersatzschalten in Kommunikationsnetzen mit Redundanzdomänen*. In: *Dissertation, Technische Universität München*, Juli 1999. 31

[JGJ+00] Jannotti, John, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek und James W. O'Toole, Jr.: *Overcast: Reliable Multicasting with an Overlay Network*. In: *Fourth Symposium on Operating Systems Design and Implementation (OSDI 2000)*, San Diego, CA, USA, Oktober 2000. 14

[JO00] Jiang, W. und A. Ortega: *Multiple description speech coding for robust communication over lossy packet networks*. In: *IEEE International Conference on Multimedia and Expo (ICME 2000)*, New York, NY, USA, August 2000. 46

[Kan01] Kan, G.: *Gnutella*. In: *In Peer-to-Peer Harnessing the Power of Diruptive Computing*, O'Reilly and Associates, März 2001. 16, 73

[Kar01] Karnouskos, Stamatis: *Dealing with Denial-of-Service Attacks in Agent-enabled Active and Programmable Infrastructures*. In: *25th IEEE International Computer Software and Applications Conference (COMPSAC 2001)*, Chicago, IL, USA, Oktober 2001. 22

[KCD+00] Keller, Ralph, Sumi Choi, Dan Decasper, Marcel Dasen, George Fankhauser und Bernhard Plattner: *An Active Router Architecture for Multicast Video Distribution*. In: *Proceedings of IEEE INFOCOM 2000*, Tel Aviv, Israel, März 2000. 47, 50, 53, 109

[KDK+02] Kuhns, Fred, John DeHart, Anshul Kantawala, Ralph Keller, John Lockwood, Prashanth Pappu, David Richards, David Taylor, Jyoti Parwatikar, Ed Spitznagel, Jon Turner und Ken Wong: *Design of a High Performance Dynamically Extensible Router*. In: *DARPA Active Networks Conference and Exposition (DANCE)*, San Francisco, Mai 2002. 51

[Kle01] Klensin, J.: *RFC 2821: Simple Mail Transfer Protocol*, April 2001. 14

[KLW01] Krasic, Charles, Kang Li und Jonathan Walpole: *The Case for Streaming Multimedia with TCP*. In: *8th International Workshop on Interactive Distributed Multimedia Systems (iDMS 2001)*, Lancaster, UK, September 2001. 44

[KMR02] Keromytis, Angelos D., Vishal Misra und Dan Rubenstein: *SOS: Secure Overlay Services*. In: *ACM SIGCOMM 2002*, August 2002. 13

[KP01] Koodli, Rajeev und Charles E. Perkins: *Fast Handovers and Context Transfers in Mobile Networks*. ACM SIGCOMM Computer Communication Review, Oktober 2001. 17

[KRGP02] Keller, Ralph, Lukas Ruf, Amir Guindehi und Bernhard Plattner: *PromethOS: A Dynamically Extensible Router Architecture Supporting Explicit Routing*. In: *4th Annual International Working Conference on Active Networks, IWAN 2002*, Zurich, Switzerland, Dezember 2002. 22, 50

[KWW+98] Knight, S., D. Weaver, D. Whipple, R. Hinden, D. Mitzel, P. Hunt, P. Higginson, M. Shand und A. Lindem: *RFC 2338: Virtual Router Redundancy Protocol (VRRP)*, April 1998. 35

[LABJ01] Labovitz, Craig, Abha Ahuja, Abhijit Bose und Farnam Jahanian: *Delayed Internet routing convergence*. In: *IEEE/ACM Transactions on Networking*, Juni 2001. 34, 35

[LCML98] Li, T., B. Cole, P. Morton und D. Li: *RFC 2281: Cisco Hot Standby Router Protocol (HSRP)*, März 1998. 35

[LJ03] Lau, William und Sanjay Jha: *APLS: An IP Label Switching Architecture*. In: *IEEE International Conference on Communications, ICC 2003*, Anchorage, Alaska, USA, Mai 2003. 42

[LOR98] Lakshman, T. V., A. Ortega und A. R. Reibman: *Variable bit-rate (VBR) video: Tradeoffs and Potentials*. Proceedings of the IEEE, 86(5), Mai 1998. 46

[LPP+01] Lefevre, L., C. Pham, P. Primet, B. Tourancheau, B. Gaidioz, J.P. Gelas und M.Maimour: *Active networking support for the grid*. In: *Proceedings of the IFIP-TC6 Third International Working Conference (IWAN'01)*, Philadelphia, PN, USA, Oktober 2001. 23

[LSG01] Liang, Yi J., E. Steinbach und B. Girod: *Real-time Voice Communication over the Internet Using Packet Path Diversity*. In: *ACM Multimedia 2001*, Ottawa, Canada, September 2001. 46

[mad00] *CNN News, Millions watch Madonna online*, November 2000. `http://www.cnn.com/2000/SHOWBIZ/Music/11/28/britain.madonna/`. 1

[MMF03] M. Marwah, S. Mishra und C. Fetzer: *TCP Server Fault Tolerance Using Connection Migration to a Backup Server.* In: *IEEE International Conference on Dependable Systems and Networks (DSN 2003)*, San Francisco, CA, Juni 2003. 44

[MMN02] Moore, Jonathan T., Jessica Kornblum Moore und Scott Nettles: *Predictable, Lightweight Management Agents.* In: *Proceedings of the IFIP-TC6 Fourth International Working Conference (IWAN'02)*, Zurich, Switzerland, Dezember 2002. 21

[Moc87a] Mockapetris, P.: *RFC 1034: Domain Names - Concepts and Facilities*, November 1987. 14

[Moc87b] Mockapetris, P.: *RFC 1035: Domain Names - Implementation and Specification*, November 1987. 14

[nap01] *The Napster Protocol*, September 2001. `http://opennap.sourceforge.net/napster.txt`. 16

[ogs03] *Open Grid Services Architecture Working Group (OGSA WG)*, August 2003. `http://www.ggf.org/ogsa-wg/`. 15

[OY02] Ong, L. und J. Yoakum: *RFC 3286: An Introduction to the Stream Control Transmission Protocol (SCTP)*, Mai 2002. 45

[Pen03] Peng, Gang: *CDN: Content Distribution Network.* In: *Research Proficiency Exam report, Computer Science Department*, SUNY at Stony Brook, NY, USA, January 2003. 40

[PSVW01] Pendarakis, Dimitris, Sherlia Shi, Dinesh Verma und Marcel Waldvogel: *ALMI: An Application Level Multicast Infrastructure.* In: *3rd USNIX Symposium on Internet Technologies and Systems (USITS '01)*, San Francisco, CA, USA, März 2001. 14

[PW02] Prehofer, Christian und Qing Wei: *Active Networks for 4G Mobile Communication: Motivation, Architecture, and Application Scenarios.* In: *4th Annual International Working Conference on Active Networks, IWAN 2002*, Zurich, Switzerland, Dezember 2002. 23

[QDF+03] Qin, Feng, Jon Dugan, Jim Ferguson, Kevin Gibbs und Ajay Tirumala: *Iperf Version 1.7.0*, März 2003. `http://dast.nlanr.net/Projects/Iperf/`. 162

[Ram00] Ramalho, M.: *Intra- and inter-domain multicast routing protocols: a survey and taxonomy.* In: *IEEE Communications, Survey and Tutorials, First Quarter 2000, Vol. 3, No. 1*, 2000. 24

[RFH+01] Ratnasamy, Sylvia, Paul Francis, Mark Handley, Richard Karp und Scott Shenker: *A Scalable Content-Addressable Network.* In: *ACM SIGCOMM*, San Diego, CA, USA, August 2001. 41

[RL95] Rekhter, Y. und T. Li: *RFC 1771: Border Gateway Protocol 4*, Juli 1995. 34

[RS99] Raz, Danny und Yuval Shavitt: *An Active Network Approach to Efficient Network Management.* In: *Proceedings of the First International Working Conference on Active Networks (IWAN '99)*, Berlin, Germany, Juni 1999. 21

[RSB01] Rodriguez, Pablo, Christian Spanner und Ernst W. Biersack: *Analysis of Web Caching Architectures: Hierarchical and Distributed Caching.* In: *IEEE/ACM Transactions on Networking*, August 2001. 40

[RWE+01] Rhea, Sean, Chris Wells, Patrick Eaton, Dennis Geels, Ben Zhao, Hakim Weatherspoon und John Kubiatowicz: *Maintenance-Free Global Data Storage.* In: *IEEE Internet Computing)*, September 2001. 41

[SAA+99] Savage, Stefan, Tom Anderson, Amit Aggarwal, David Becker, Neal Cardwell, Andy Collins, Eric Hoffman, John Snell, Amin Vahdat, Geoff Voelker und John Zahorjan: *Detour: Informed Internet Routing and Transport.* In: *IEEE Micro, v 19, no 1*, Januar 1999. 36, 42

[Sch01] Schollmeier, R.: *A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications.* In: *IEEE International Conference on Peer-to-Peer Computing, P2P'01*, Linkoeping, Sweden, August 2001. 15

[SGPW02] Shalaby, N., Y. Gottlieb, L. Peterson und M. Wawrzoniak: *Snow on Silk: A NodeOS in the Linux kernel.* In: *4th Annual International Working Conference on Active Networks, IWAN 2002*, Zurich, Switzerland, Dezember 2002. 50

[SH03] Schollmeier, R. und F. Hermann: *Topology-analysis of pure Peer-to-Peer networks.* In: *Kommunikation in Verteilten Systemen KiVS 2003*, Leipzig, Germany, Februar 2003. 16

[Sim95] Simpson, W.: *RFC 1853: IP in IP Tunneling*, Oktober 1995. 12, 13, 42

[SKS00] Schäfer, R., P. Kauff und O. Schreer: *Immersive Telepresence for Broadcast and Communication.* In: *Proc. of IST (Information Society Technologies) 2000 Event*, Nice, France, November 2000. 1

[SLG02] Steinbach, E., Y. Liang und B. Girod: *A Simulation Study of Packet Path Diversity for TCP File Transfer and Media Transport on the Internet.* In: *Tyrrhenian International Workshop on Digital Communications (IWDC 2002)*, Capri, Italy, September 2002. 44

[SMK+01] Stoica, I., R. Morris, D. Karger, M. F. Kaashoek und H. Balakrishnan: *Chord: A scalable peer-to-peer lookup service for Internet applications.* In: *MIT Technical Report TR-819*, Cambridge, MA, USA, März 2001. 16

[SP82] Su, Zaw-Sing und Jon Postel: *RFC 819: The Domain Naming Convention for Internet User Applications*, August 1982. 14

[spe04] *Speedera Inc. SpeedSuite*, Januar 2004. `http://www.speedera.com/primary/services/speedsuite.htm`. 39

[SRBW01] Schläger, M., B. Rathke, S. Bodenstein und A. Wolisz: *Advocating a Remote Socket Architecture for Internet Access using Wireless LANs*. Mobile Networks & Applications (Special Issue on Wireless Internet and Intranet Access), Januar 2001. 21, 54

[SS02] Schollmeier, R. und G. Schollmeier: *Why Peer-to-Peer (P2P) does scale: an analysis of P2P traffic patterns*. In: *IEEE International Conference on Peer-to-Peer Computing, P2P'02*, Linkoeping, Sweden, September 2002. 16

[SSBK04] Subramanian, Lakshminarayanan, Ion Stoica, Hari Balakrishnan und Randy Katz: *OverQoS: An Overlay based Architecture for Enhancing Internet QoS*. In: *First Symposium on Networked Systems Design and Implementation (NSDI'04)*, San Francisco, CA, USA, März 2004. 62

[ST01] Seiichiro Tani, Toshiaki Miyazaki, Noriyuki Takahashi: *Adaptive Stream Multicast Based on IP Unicast and Dynamic Commercial Attachment Mechanism: an Active Network Implementation*. In: *Proceedings of the IFIP-TC6 Third International Working Conference (IWAN'01)*, Philadelphia, PN, USA, Oktober 2001. 24

[TB02] Tabery, Peter und Christian Bachmeir: *Advanced Network Services using Programmable Networks*. In: *IFIP Workshop and EUNICE Summer School on Adaptable Networks and Teleservices, Trondheim, Norway*, September 2002. 32, 145, 147

[TBW04] Tabery, Peter, Christian Bachmeir und Nicolas Wimmer: *Introducing Handoff-Triggered TCP (hot-TCP) for Throughput-Optimized Mobility Support*. In: *World Multi-Conference on Systemics, Cybernetics and Informatics, SCI'04*, Orlando, Fl, USA, Juli 2004. 122, 145

[TG04] Turrini, E. und V. Ghini: *A Protocol for Exchanging Performance Data in Content Distribution Internetworks*. In: *3rd IEEE International Conference on Networking, (ICN'04)*, Gosier, Guadeloupe, März 2004. 107

[Tou01] Touch, Joe: *Dynamic Internet Overlay Deployment and Management Using the X-Bone*. In: *IEEE Computer Networks*, Juli 2001. 12

[Tsa98] Tsakiridou, Evdoxia: *Sechs Sterne für gute Software*. In: *Forschung und Innovation, Siemens-Zeitschrift*, Januar 1998. 24

[voi04] *VoIP, Federal Communications Commission*, Mai 2004. `http://www.fcc.gov/voip/`. 1

[web04] *W3C, Web Services Activity*, Januar 2004. `http://www.w3.org/2002/ws/`. 121

[WGT98] Wetherall, D., J. Guttag und D. Tennenhouse: *ANTS: A toolkit for building and dynamically deploying network protocols*. In: *IEEE OPENARCH'98*, San Francisco, CA, USA, Oktober 1998. 18, 50

[WLG98] Wetherall, David, Ulana Legedza und J. Guttag: *Introducing New Internet Services: Why and How.* In: *IEEE Network*, Mai 1998. 18

[WPD88] Waitzman, D., C. Partridge und S. Deering: *RFC 1075: Distance Vector Multicast Routing Protocol*, November 1988. 12

[Zha02] Zhao, Xiaoliang: *Toward a Fault-Tolerant Border Gateway Protocol.* In: *Ph.D. Thesis, NCSU*, 2002. 34

[ZHS+03a] Zhao, Ben Y., Ling Huang, Jeremy Stribling, Anthony D. Joseph und John Kubiatowicz: *Exploiting Routing Redundancy via Structured Peer-to-Peer Overlays.* In: *11th International Conference on Network Protocols*, November 2003. 38

[ZHS+03b] Zhao, Ben Y., Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph und John D. Kubiatowicz: *Tapestry: A Resilient Global-scale Overlay for Service Deployment.* In: *IEEE Journal on Selected Areas in Communications*, November 2003. 38, 41, 42

[Zim99] Zimmer, Tobias: *Virtuelle private Netze - weltweite LANs.* In: *Technical Report: Netzwerk-Management und Hochleistungs-Kommunikation, ISSN 1432-7864*, April 1999. 13

Vorveröffentlichungen gemäß
Promotionsordnung der Technischen Universität München § 6 Abs. 1:
[TB02, BT02, Bac03, BTK03, BPV+03, BT03, BTS+03, BTM+04].

Weitere Veröffentlichungen des Autors: [BTÜS03, TBW04].

Abbildungsverzeichnis

Anhang A

Evaluierung zur Wege- und Contentfindung

Alternative Wege

In diesem Abschnitt beschreiben wir die Ergebnisse von Messungen im Internet. Dabei werden *direkte Pfade* vom Standort der TU München ausgehend, zu über den Globus verteilten Zielrechnern, mit *alternativen Pfaden* verglichen. Wie in Kapitel 5 angeführt, stützen sich die alternativen Pfade dabei jeweils nur auf ein Relay.

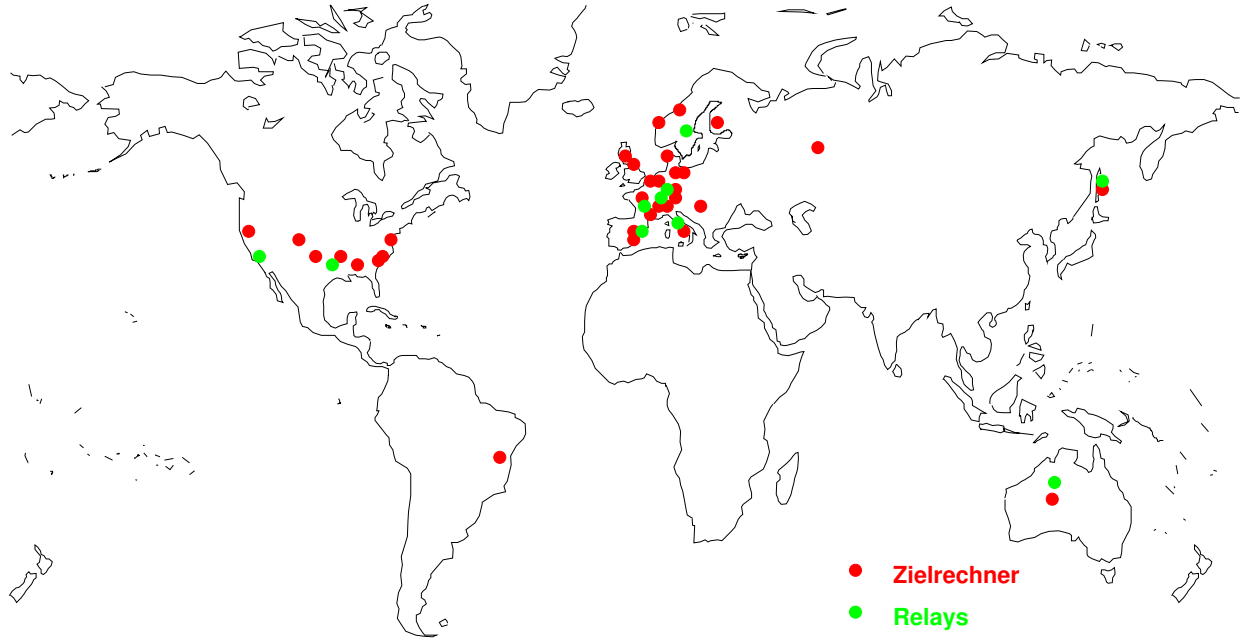

Abbildung A.1: Geographische Verteilung

In Abbildung A.1 ist die Verteilung der Zielrechner und der Relays skizziert. Als Ziele dienen reguläre, mit dem Internet verbundene Rechner. Die Relays werden in den Messungen durch erweiterte Server zur Verfügung gestellt, die eine entfernte Traceroute [BBCW01] erlauben.

Zu den Zielrechnern und Relays werden jeweils vom Standort der TU München ausgeführte, direkte Traceroutes durchgeführt. Ausgehend von den beteiligten Relays werden ebenfalls Traceroutes zu den Zielen durchgeführt. Die abschnittsweisen Traceroutes, vom Standort zu den Relays und von den Relays zu den Zielen, setzen wir zu jeweils einer Traceroute zusammen.
Anschließend werden die direkten Traceroutes zu den Zielen (TU München ⇒ Zielrechner), mit den zusammengesetzten Traceroutes (TU München ⇒ Relay ⇒ Zielrechner) verglichen.

Distanz (ms) / Relays / Ziele	22	25	30	32	41	101	321	390	178	133
	www.informatik.rwth-aachen.de	www.univ-st-etienne.fr	cgi.student.nada.kth.se	www.unige.ch	trace.cere.pa.cnr.it	web.ocea.es	web.planet.co.jp	www.telstra.net	www.sdsc.edu	taex001.tamu.edu
www.murdoch.edu.au						X	X	X		X
www.ulg.ac.be	X	X		X		X	X	X		
www.uerj.br					X	X	X	X	X	
www.cs.tut.fi						X	X	X		
www.enst.fr	X	X	X	X	X	X	X	X	X	X
www.uni-karlsruhe.de			X	X	X	X	X	X		
www.uni-stuttgart.de			X	X	X	X	X			
www.bme.hu						X	X	X		
www.polio.it						X	X	X		
www.utwente.nl						X	X	X		
www.unik.no					X	X	X	X		
www.ntnu.no		X				X	X	X		
www.sut.ru						X	X	X		
www.upc.es		X				X	X	X		
www.upm.es		X				X	X	X		
www.epfl.ch		X			X		X	X		

Distanz (ms) / Relays / Ziele	22	25	30	32	41	101	321	390	178	133
	www.informatik.rwth-aachen.de	www.univ-st-etienne.fr	cgi.student.nada.kth.se	www.unige.ch	trace.cere.pa.cnr.it	web.ocea.es	web.planet.co.jp	www.telstra.net	www.sdsc.edu	taex001.tamu.edu
www.ucl.ac.uk			X			X	X	X		
www.colorado.edu						X	X	X		
www.uiuc.edu					X	X	X	X		
www.utexas.edu						X	X	X		
www.scr.siemens.com		X	X	X		X	X	X	X	X
www.princeton.edu						X	X	X		
www.stanford.edu						X	X	X	X	X
sloan.mit.edu						X	X	X		
www.ee.ucl.ac.uk			X			X	X	X		
www.ijs.si						X	X	X		
www.berkom.de	X	X	X	X		X	X	X	X	X
www.knp.com		X	X			X	X	X	X	X
www.hitachi.co.jp		X	X	X		X	X	X	X	X
www.ethz.ch		X	X			X	X	X		
www.ikv.de						X	X	X	X	X
www.upenn.edu						X	X	X		

Abbildung A.2: Alternative Pfade

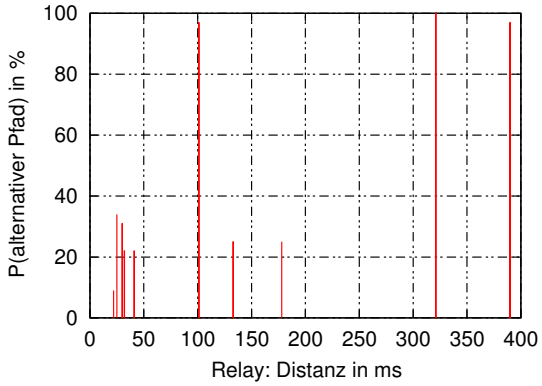

Abbildung A.3: Wahrscheinlichkeit der prinzipiellen Bereitstellung eines alternativen Pfads

In den Abbildungen A.2 und A.3 präsentieren wir die Auswertung der Messungen. Es zeigt sich, dass für alle Ziele, mehrere alternative Pfade durch das Internet bereitgestellt werden können. Das bereits im RON-Projekt [ABKM01] gefundene, empirische Ergebnis konnte somit bestätigt werden. Im Unterschied zu RON, sind in unserer Messung aber auch Ziele bzw. Zielrechner und Relays ausserhalb der USA mit einbezogen.
Des Weiteren zeigen die Messungen, dass größere Distanzen zu den Relays die Bereitstellung eines alternativen Pfades begünstigen. Dieses Prinzip nehmen wir in den Aufbau eines Clusters auf, wie in Abschnitt 5.3 beschrieben. Die in Abschnitt 5.3.3 beschriebene Aufnahme eines Connecting-Peers in einen Cluster berücksichtigt die Distanzen innerhalb des Clusters. Somit ist unserer Ansicht nach gewährleistet, dass für jeden Connecting-Peer einige Relays zur Verfügung stehen, die alternative Pfade zur Verfügung stellen können.
Darüber hinaus verhindert der Mechanismus eine topologische Konzentration von Teilnehmern eines Cluster. Kommt es zu Ausfällen in Teilbereichen des Netzes, ist es somit unwahrscheinlich, dass gleichzeitig mehrere Relays eines Clusters nicht zur Verfügung stehen.

Ferner zeigte sich in den, über einen Zeitraum von zwei Wochen, durchgeführten Messungen, dass sich Pfade durch das Internet im Lauf der Zeit nicht häufig ändern [Bac03]. Dies ist eine weitere Grundlage für die in Kapitel 5 vorgestellten Overlay-Netze zur Wegefindung:
Im übergeordneten P2P-Netz werden kontinuierlich Pfade zwischen den beteiligten Connecting-Peers mittels Traceroute bestimmt (siehe Abschnitt 5.2). Diese Messungen bilden die Basis für den in Abschnitt 5.3 beschriebenen Aufbau der Cluster. Da sich die Pfade nicht häufig ändern, kann auch angenommen werden, dass sich die Güte der jeweiligen Cluster nicht oft verändern wird.
Die in Abschnitt 5.3.8 präsentierten dynamischen RON, führen nicht für jede neu zu schützende Datenverbindung Messungen zur Evaluierung eines alternativen Pfades durch. Sie bauen ebenfalls auf den bereits zur Verfügung stehenden Daten aus dem übergeordneten P2P-Netz auf. Diese Daten sind, durch ein Zeitintervall im höherschichtigen P2P-Netz bedingt, bis zu ca. einer Stunde alt. Auch hierfür gilt daher, dass die zur Verfügung stehenden Daten eine gute Basis für den Mechanismus sind.

In den Abbildungen A.4 bis A.35 präsentieren wir einzelne Messungen im Detail:

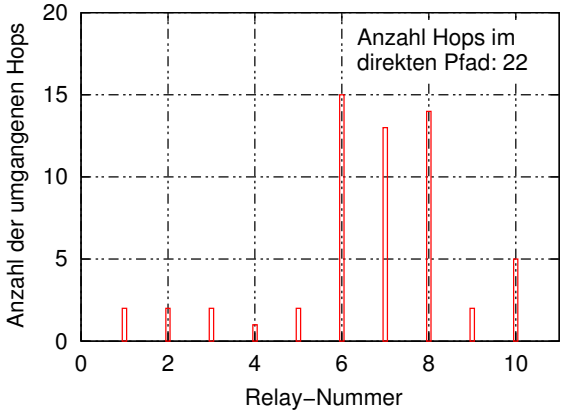

Abbildung A.4: www.murdoch.edu.au: Umgangene Hops

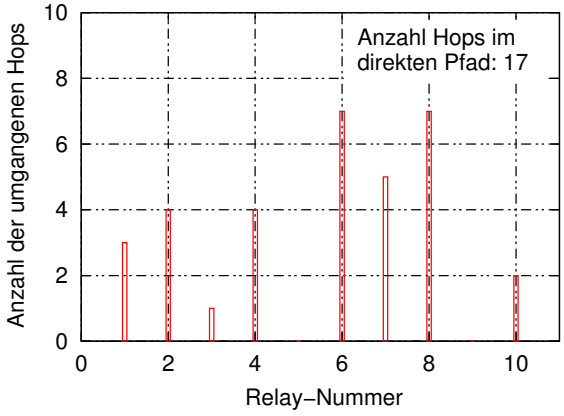

Abbildung A.5: www.ulg.ac.be: Umgangene Hops

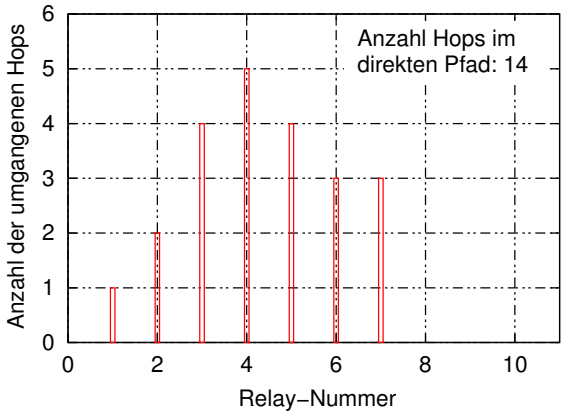

Abbildung A.6: www.uni-stuttgart.de: Umgangene Hops

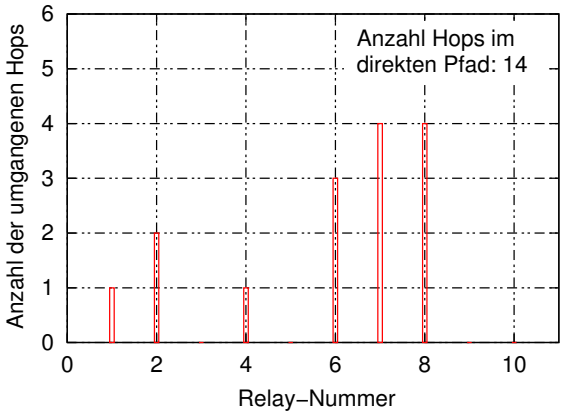

Abbildung A.7: www.bme.hu: Umgangene Hops

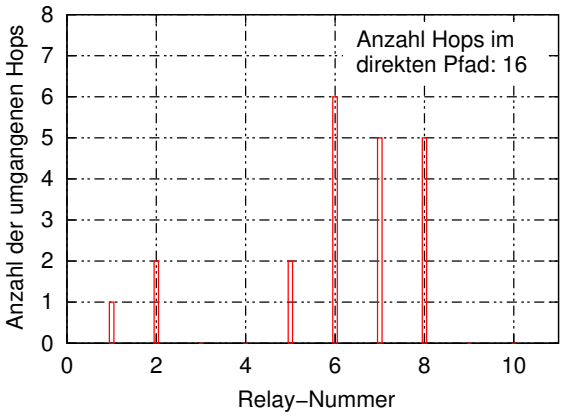

Abbildung A.8: www.polito.it: Umgangene Hops

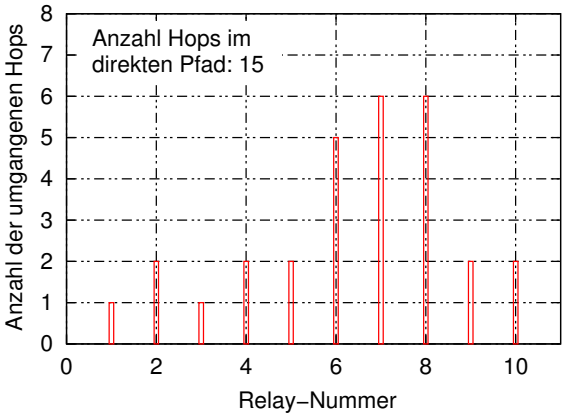

Abbildung A.9: www.utwente.nl: Umgangene Hops

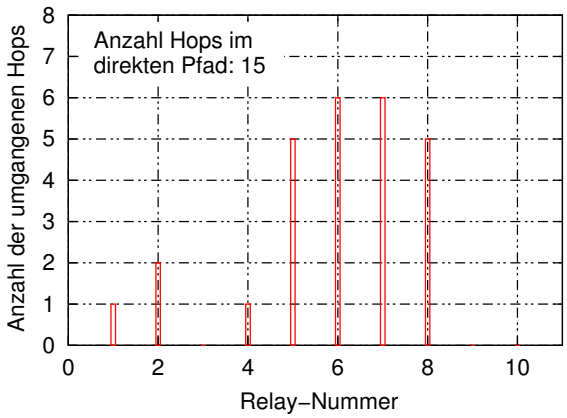

Abbildung A.10: www.unik.no: Umgangene Hops

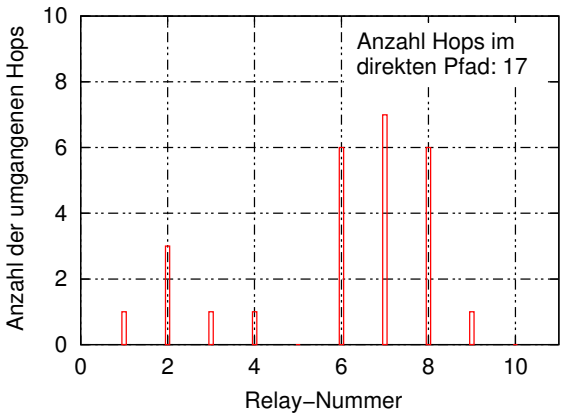

Abbildung A.11: www.ntnu.no: Umgangene Hops

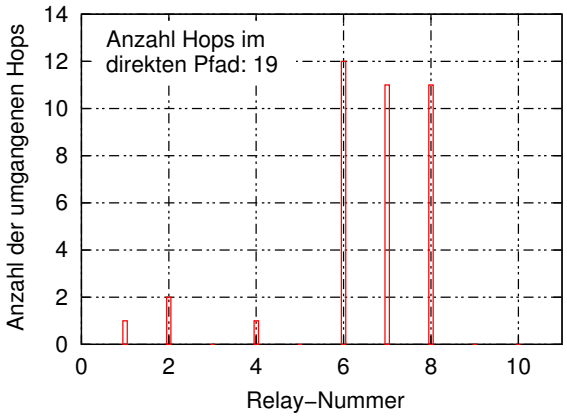

Abbildung A.12: www.sut.ru: Umgangene Hops

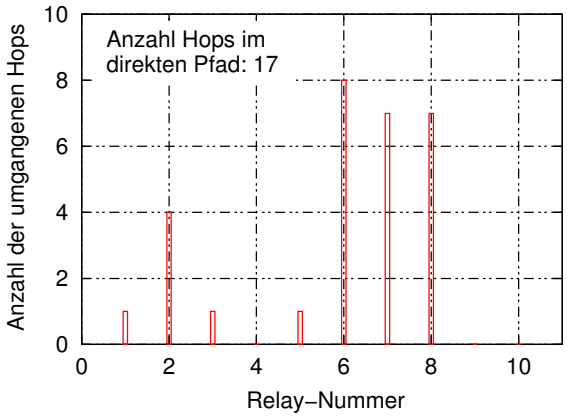

Abbildung A.13: www.upc.es: Umgangene Hops

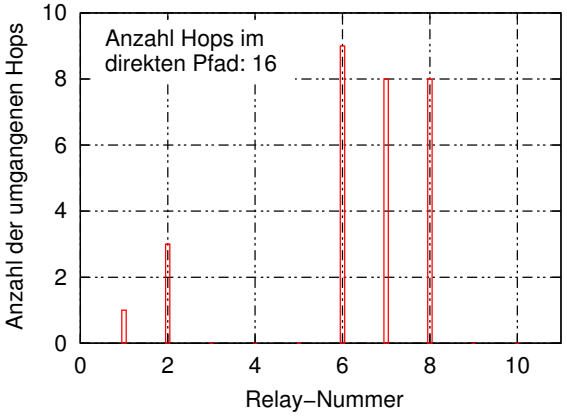

Abbildung A.14: www.upm.es: Umgangene Hops

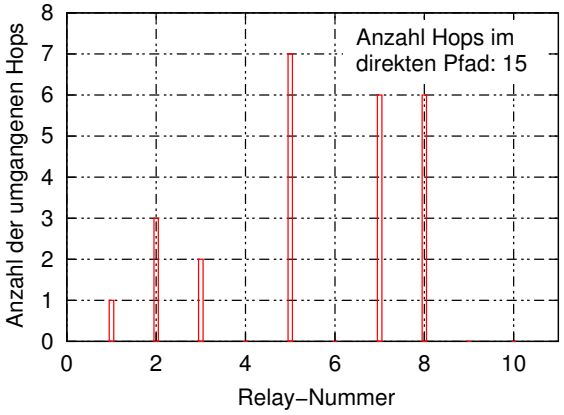

Abbildung A.15: www.epfl.ch: Umgangene Hops

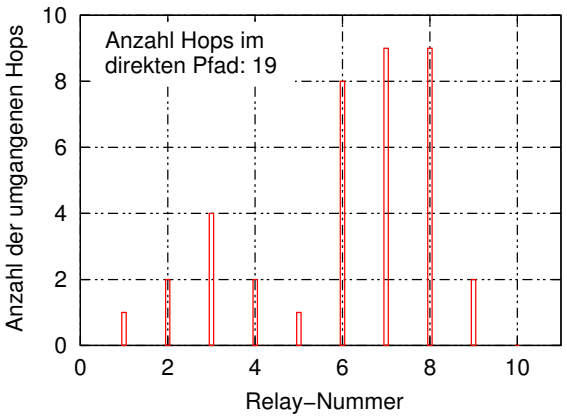

Abbildung A.16: www.ucl.ac.uk: Umgangene Hops

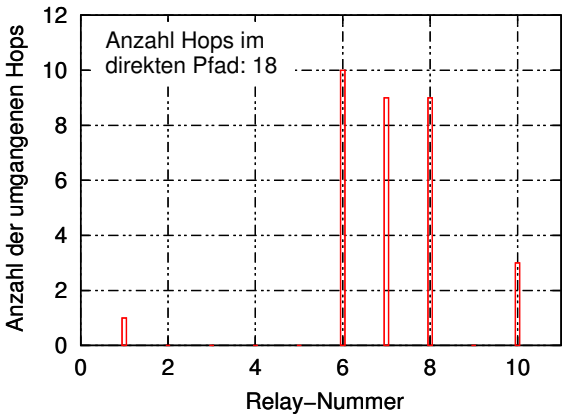

Abbildung A.17: www.colorado.edu: Umgangene Hops

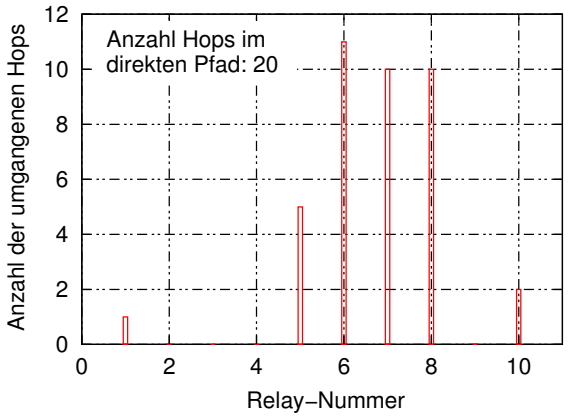

Abbildung A.18: www.uiuc.edu: Umgangene Hops

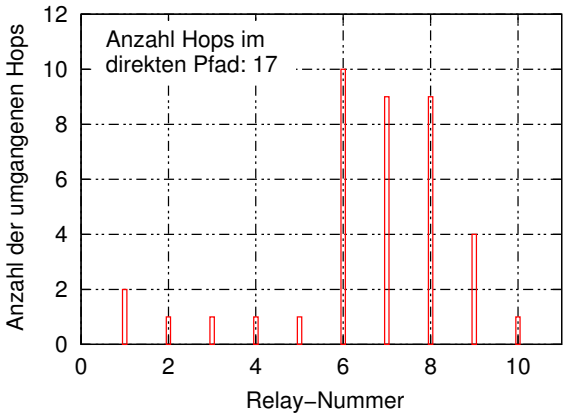

Abbildung A.19: www.utexas.edu: Umgangene Hops

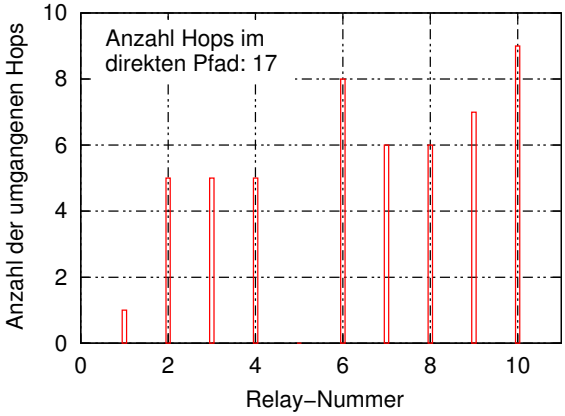

Abbildung A.20: www.scr.siemens.com: Umgangene Hops

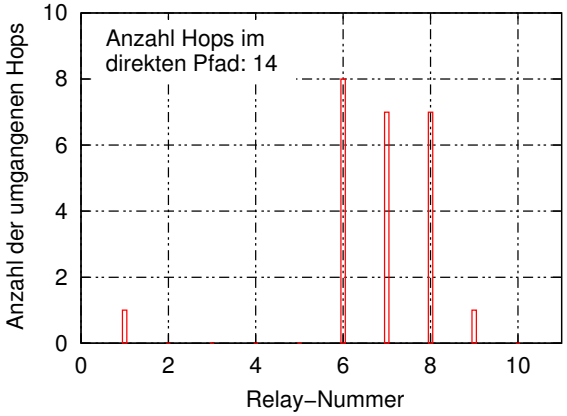

Abbildung A.21: www.princeton.edu: Umgangene Hops

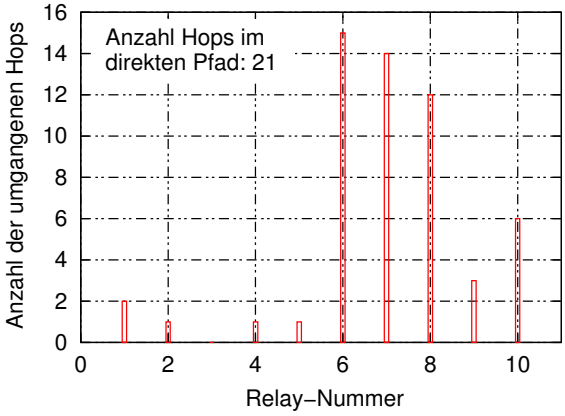

Abbildung A.22: www.stanford.edu: Umgangene Hops

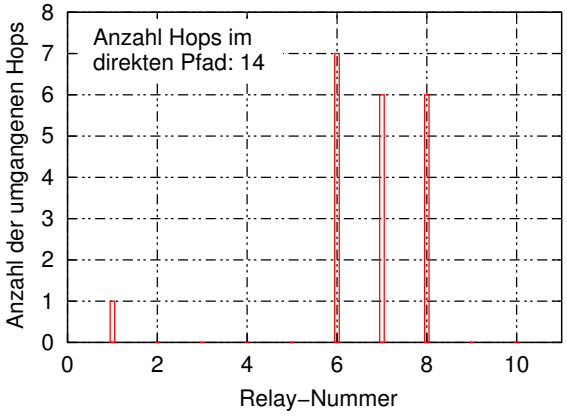

Abbildung A.23: sloan.mit.edu: Umgangene Hops

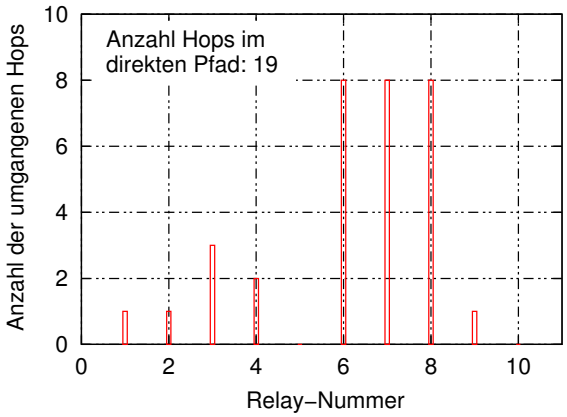

Abbildung A.24: www.ee.ucl.ac.uk: Umgangene Hops

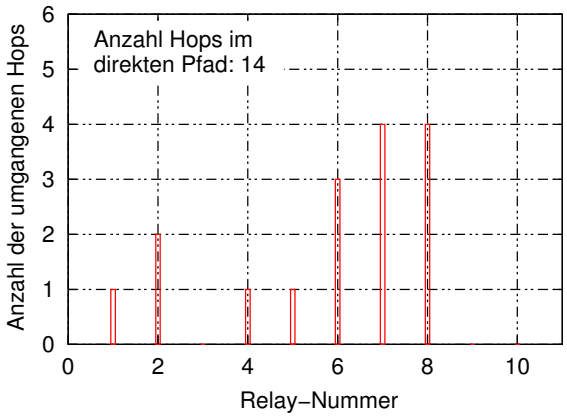

Abbildung A.25: www.ijs.si: Umgangene Hops

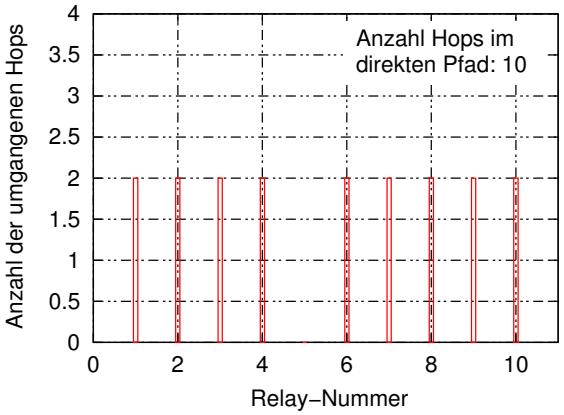

Abbildung A.26: www.berkom.de: Umgangene Hops

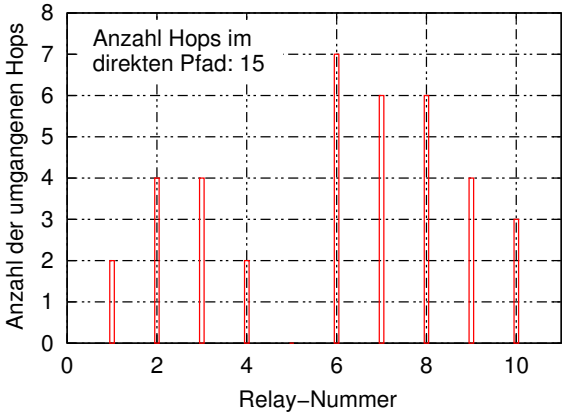

Abbildung A.27: www.kpn.com: Umgangene Hops

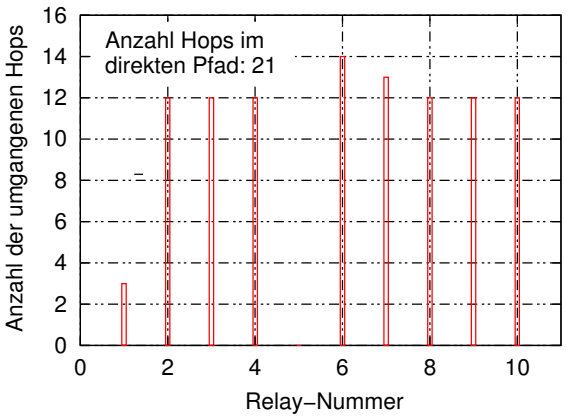

Abbildung A.28: www.hitachi.co.jp: Umgangene Hops

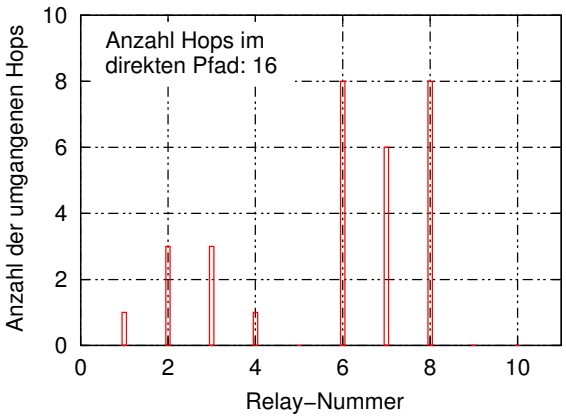

Abbildung A.29: www.ethz.ch: Umgangene Hops

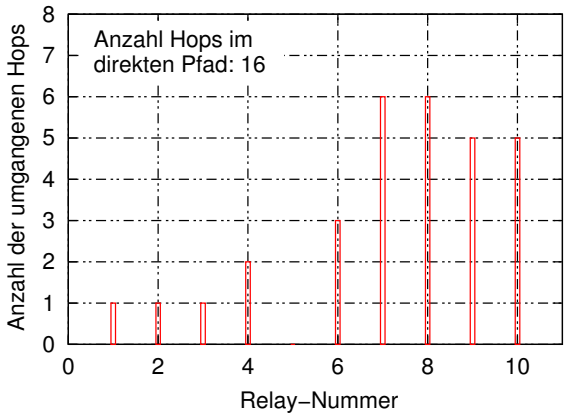

Abbildung A.30: www.ikv.de: Umgangene Hops

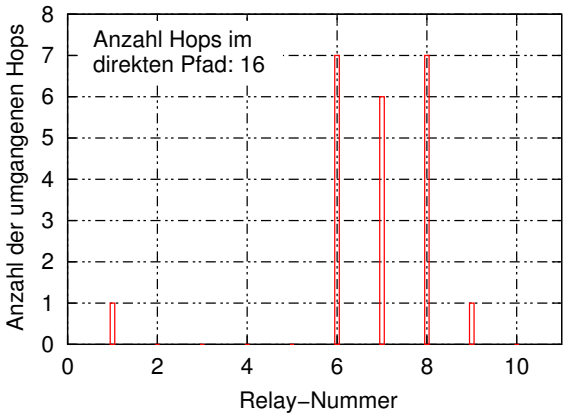

Abbildung A.31: www.upenn.edu: Umgangene Hops

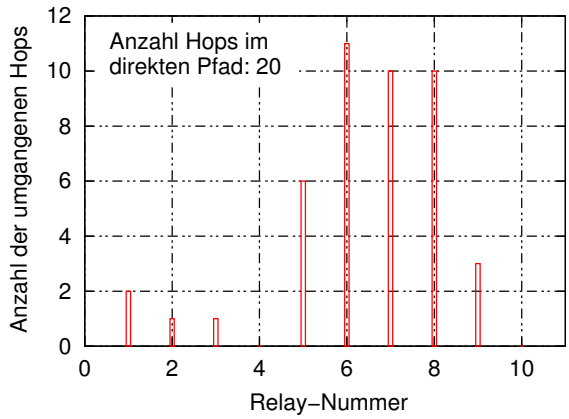

Abbildung A.32: www.uerj.br: Umgangene Hops

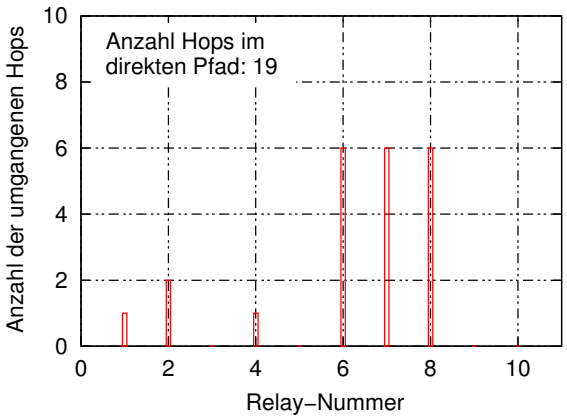

Abbildung A.33: www.cs.tut.fi: Umgangene Hops

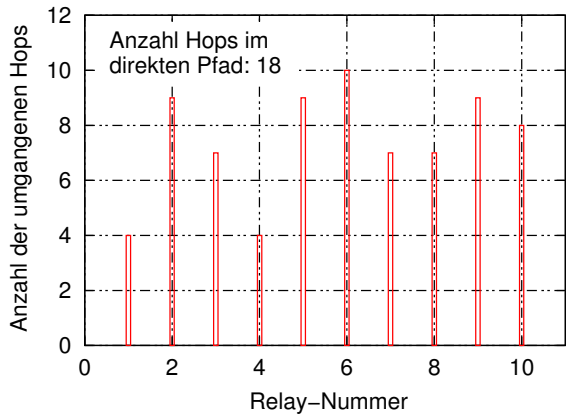

Abbildung A.34: www.enst.fr: Umgangene Hops

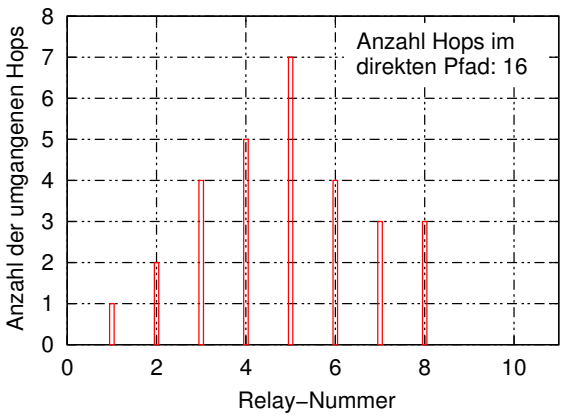

Abbildung A.35: www.uni-karlsruhe.de: Umgangene Hops

Kooperierende Cache-Cluster

In Abbildung A.36 sind exemplarisch einige Cache-Cluster innerhalb einer administrativen Domäne dargestellt. Innerhalb der Cache-Cluster sind Zugangsknoten über einen Multicast-Baum miteinander verbunden und tauschen ihre Cache-Summaries aus, wie in Abschnitt 5.4 beschrieben.
Wir gehen von der in Abschnitt 5.3.9 getroffenen Annahme aus. Innerhalb einer administrativen Domäne sind maximal 256 Zugangsknoten beinhaltet. Diese schließen sich zu Cache-Clustern zusammen mit jeweils ca. 10 teilnehmenden Zugangsknoten. Daher sind maximal 26 Cache-Cluster innerhalb einer administrativen Domäne zu erwarten.
Zugangsknoten bilden Overlay-Netze mit anderen Zugangsknoten aus allen Cache-Clustern in einer Domäne. Bei der Bildung der Overlay-Netze ist darauf zu achten, dass in jedem Overlay-Netz nur jeweils ein Zugangsknoten eines Cache-Clusters vertreten ist. Somit entspricht die Anzahl der Overlay-Netze der Zahl der Zugangsknoten in den Cache-Clustern.

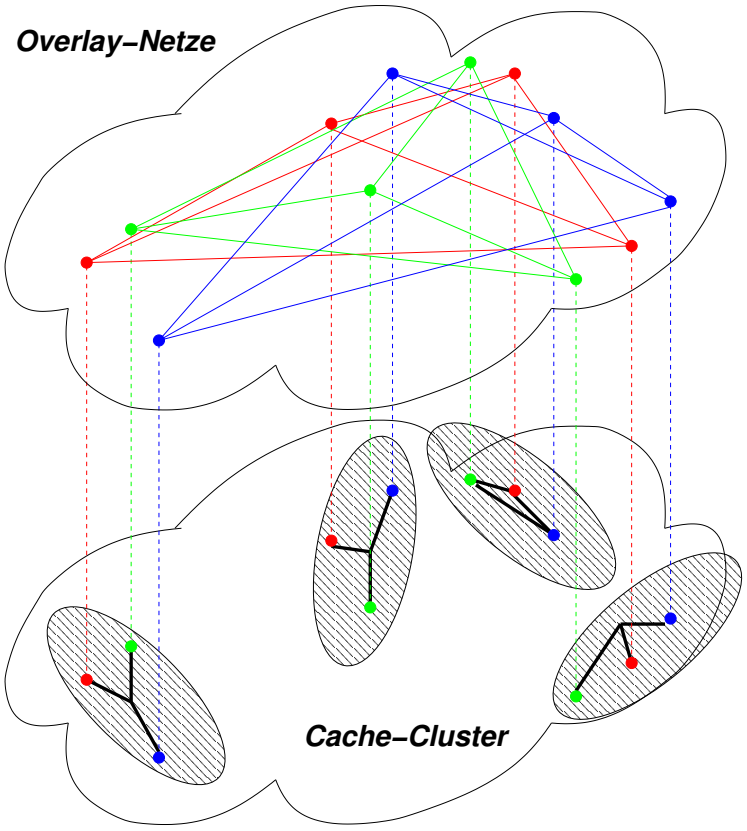

Abbildung A.36: Overlay-Netze zwischen Cache-Clustern

Daraus kann der Signalisierungsaufwand für die jeweiligen Overlay-Netze folgendermaßen abgeschätzt werden: Über ein Overlay-Netz sind maximal 26 Zugangsknoten verbunden. Wir gehen davon aus, wie in Abschnitt 5.4.3 beschrieben, dass die longitudinale Fehlermaskierung nur in Kraft tritt, wenn ein Contentausfall vorliegt. Weiter gehen wir für einen Zugangsknoten von der Annahme von in etwa 100 fehlschlagenden Verbindungen pro Stunde aus, bedingt durch Contenausfall. Für diese Verbindungen tritt die beschriebene longitudinale Fehlermaskierung in Kraft.

Nehmen wir weiter den ungünstigsten Fall an, dass diese Verbindungen nicht durch vorliegenden Content aus dem Cache-Cluster eines Zugangsknoten bedient werden können, dann versucht der Fehlertoleranzdienst den gesuchten Content innerhalb seines Overlay-Netzes in den anderen Cache-Clustern seiner Domäne zu finden. Somit ergibt sich für einen Zugangsknoten ein Signalisierungsausfwand von ca. 1,2 KBits*.
Aufgrund der angenommenen kleinen Anzahl an fehlschlagenden Verbindungen, durch Contentausfall bedingt, ist auch die zu erwartende benötigte Übertragungsrate für den anschließenden Transfer von gefundenen Dokumenten gering.

*100 (eigene Anfragen / pro Stunde) $*$ 200 Byte $*$ 26 $\approx$ 1,2 KBit/s

Anhang B

Evaluierung im Testnetz

SNIPLS: Vergleich mit anderen Verfahren

Im Folgenden beschreiben wir eine Messung zur Evaluierung des in Abschnitt 5.5 beschriebenen *Sparse Networkspanning IP Label Switching* (SNIPLS). Wir transferieren eine Datei mit dem TCP/IP-Protokoll zwischen zwei Rechnern in einem Testnetz. Dazu benutzen wir reguläre TCP-Sockets auf den Maschinen.

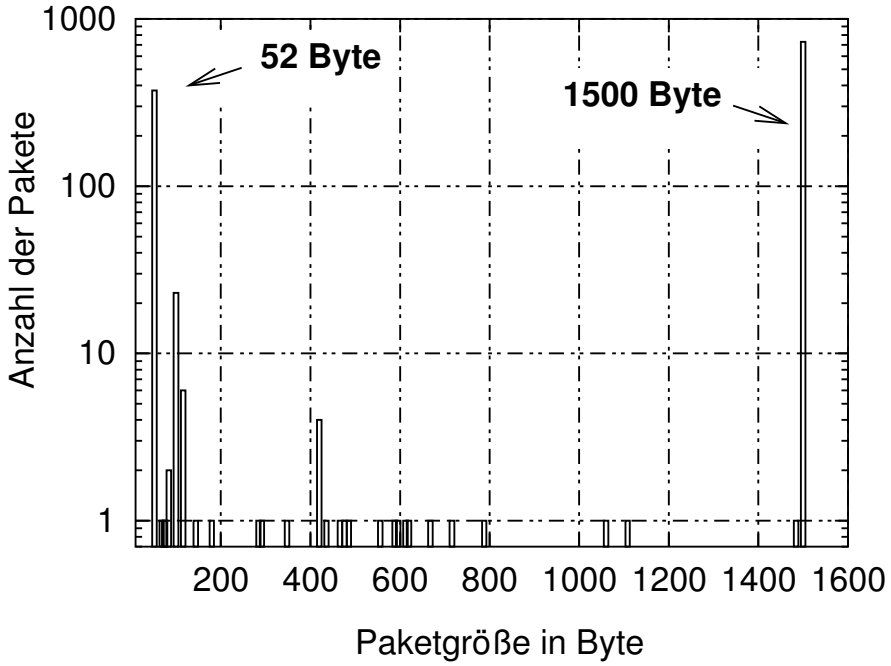

Abbildung B.1: Verteilung der Paketgröße in einer TCP-Verbindung

In Abbildung B.1 ist die Verteilung der Größe der IP-Pakete für diese Verbindung angegeben. Die Größe der übertragenen Datei beträgt **1.085.124 Byte**.
Durch das TCP/IP-Protokoll werden insgesamt **1161** IP-Pakete zwischen den Rechnern

ausgetauscht. Die insgesamt auf der Internetschicht ausgetauschte Datenmenge beläuft sich auf **1.129.941 Bytes**. Der durch TCP/IP induzierte Overhead beläuft sich somit auf **44817 Byte** bzw. auf ca. 4 % der Nutzdaten.
In der Verteilung der Paketgröße lassen sich zwei Spitzen identifizieren:

- **Lange Original-IP-Pakete**: 1500 Byte; durch MTU festgelegt
- **Kurze Original-IP-Pakete**: 52 Byte; TCP/IP-Acknowledgement

Diese entsprechen den in Abschnitt 3.4.2 identifizierten Größen von Datenpaketen.

Im Folgenden wiederholen wir die Messung für den Fall des Datentransports in einem Overlay-Netz: Wir gehen davon aus, dass ein allgemeines, transparentes 1+1 Ersatzschalten erfolgen soll, mit einem der in Abschnitt 3.4.1 beschriebenen Verfahren des Datentransports in einem Overlay-Netz.
In Abbildung B.2 zeigen wir die Verteilung der Paketgröße, wenn die in Abbildung B.1 dargestellten IP-Pakete in einem Standard-Overlay-Netz transportiert werden. Dabei nehmen wir für die zusätzlichen Header eine Größe von 20 Byte an, wie dies z.B. für IP-in-IP-Enkapsulierung der Fall ist.

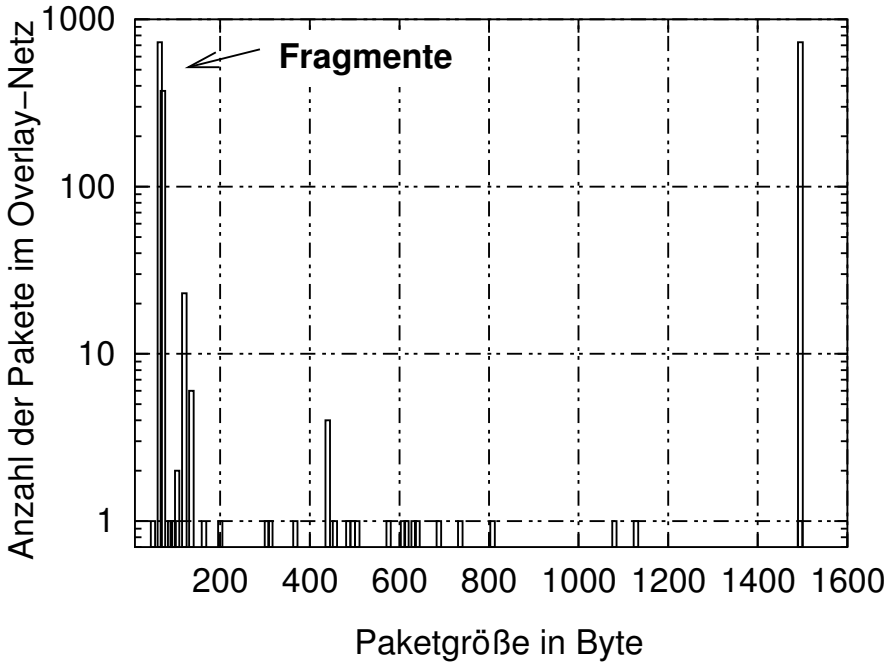

Abbildung B.2: Verteilung der Paketgröße: Enkapsulierte Pakete

Die Annahme des transparenten Datentransports durch das Overlay-Netz führt dazu, dass alle Datenpakete aus Abbildung B.1, deren Größe über 1480 Byte liegt, fragmentiert werden müssen. Daher sehen wir in Abbildung B.2 eine zusätzliche Spitze bei der Verteilung der Paketgröße, die durch Fragmente von Datenpakten begründet ist. Insgesamt werden im Fall der Fragmentierung **1891** Datenpakete durch das Overlay-Netz transportiert. Das

Datenaufkommen auf der Internschicht beläuft sich auf **1182361 Byte**. Im Vergleich zur Größe der transferierten Datei ergibt sich somit ein durch TCP/IP und das Overlay-Netz induzierter Overhead von ca. 9 %.

Unabhängig davon, ob Fragmente entstehen oder nicht, wird allein dadurch, dass ein zusätzlicher Header für den Datentransport in Overlay-Netzen* benötigt wird, Overhead übertragen.
In unserem Beispiel sind das, abhängig vom angewandten Verfahren, ca. 5 % an Daten auf der Internetschicht.

Diese Ineffizienz umgeht das in Abschnitt 5.5 vorgeschlagene SNIPLS. Unser Mechanismus benötigt keine zusätzlichen, physikalischen Header für den Datentransport in Overlay-Netzen. Bereits in den IP-Paketen vorhandene Felder werden als virtuelle Label genutzt. Durch iterative Manipulation dieser virtuellen Label erfolgt der Transport im Overlay-Netz. Somit wird die Größe der einzelnen Pakete nicht verändert. Die Verteilung der Größe der Datenpakete, in einem auf SNIPLS aufbauenden Overlay-Netz, entspricht exakt der Verteilung des originalen Datenstroms in Abbildung B.1.

Deshalb ist SNIPLS im Vergleich zu den in Abschnitt 3.4.1 präsentierten Standardverfahren effizienter. Darüber hinaus ermöglicht SNIPLS vollständige Transparenz beim Datentransport in Overlay-Netzen gegenüber beteiligten Endsystemen. Eine Anpassung der MTU (bzw. Maximum Segment Size, MSS) in den Endsystemen, wie dies z.B. bei RON geschieht, ist nicht erforderlich.

Durchsatz der komponentenbasierten programmierbaren Knotenarchitektur

In diesem Abschnitt gehen wir auf die prototypisch evaluierte Performanz der in Kapitel 6 dargestellten komponentenbasierten, programmierbaren Knotenarchitektur ein. In Abbildung B.3 ist der gemessene Paketverlust in unserem Prototypen dargestellt.

In Anlehnung an die Darstellung in Abschnitt 6.2.1, besteht unser Prototyp aus fünf Rechnern. Ein 700 MHz Pentium III PC repräsentiert den erweiterten Router. Das Enkapsulierungs-/Dekapsulierungsmodul ist als Kern-Modul in Linux realisiert. Der Rechner verfügt über vier GBit-Netzwerkkarten und fungiert somit als echter Router.
Die anderen beteiligten Rechner sind Pentium-4-basierte Standardrechner. Zwei dieser Rechner, ein Steuerungsrechner und ein Prozessrechner, sind direkt mit dem Router verbunden. Die anderen beiden Rechner, Client und Server, sind über Fast-Ethernet mit dem Router verbunden.

In unseren Messungen senden wir Datenströme vom Server zum Client. Beim Passieren des Routers werden die Datenpakete enkapsuliert und an den Prozessrechner vermittelt. Dort empfängt ein exemplarischer Dienst die Datenpakete über UDP-Sockets. Eine Verarbeitung der Datenpakete entfällt. Die empfangenen Datenpakete werden sofort nach dem Empfang

*Wenn die Standardverfahren aus Abschnitt 3.4.1 zum Einsatz kommen.

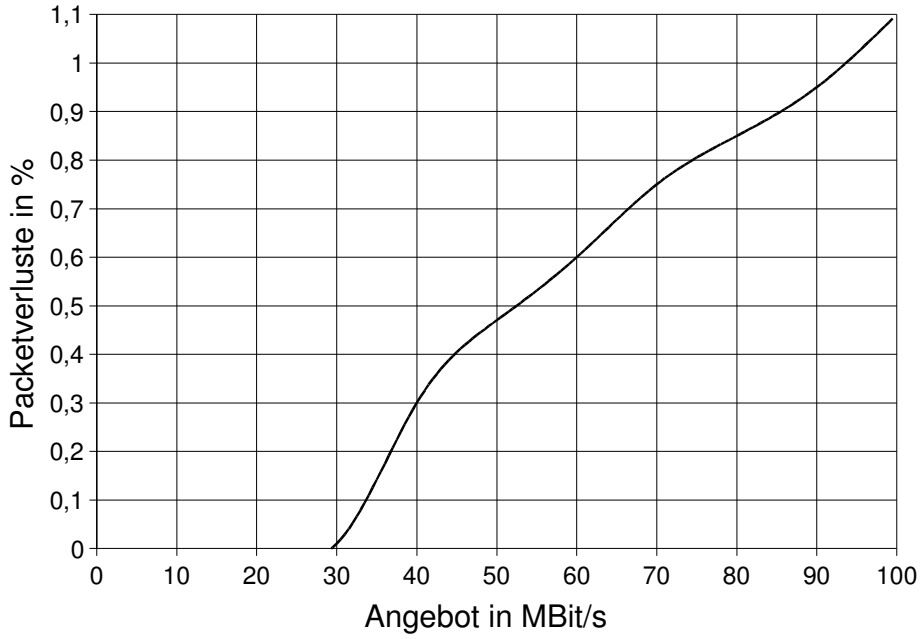

Abbildung B.3: Paketverlust im Prototypen

wieder über UDP-Sockets enkapsuliert und an den Router zurückgeschickt. Dieser dekapsuliert die Datenpakete und sendet sie an den Client.
Auf Client und Server wurde, mittels Iperf-Software [QDF+03], der Durchsatz durch den komponentenbasierten, programmierbaren Knoten ermittelt.

Obwohl die im Prototyp für den Router benutze Hardware veraltet ist, wurde nur ein sehr geringer Paketverlust festgestellt. Bemerkenswert ist das bis zu einem Angebot von ca. 30 MBit/s keine Verluste auftreten. Selbst wenn der angebotene Verkehr vom Server gegen 100 MBit/s geht, gehen durch die Enkapsulierung und Dekapsulierung nur ca. 1,1 % der Datenpakete auf dem Weg zum Client verloren.
Vergleicht man diesen Wert mit Messungen von Single Machine-Architekturen (wie z.B. in [CSF+03]), ist unser Ansatz der komponentenbasierten, programmierbaren Knotenarchitektur somit klar wettbewerbsfähig.