Simulation komplexer Cache-Verbünde im World Wide Web

Vom Fachbereich Elektrotechnik und Informationstechnik

der Universität Hannover

zur Erlangung des akademischen Grades

Doktor-Ingenieur

genehmigte

Dissertation

von

Dipl.-Ing. Christian Grimm

geboren am 12. Juli 1966 in Minden

2002

1. Referent	Prof. Dr.-Ing. Helmut Pralle
2. Referent	Prof. Dr.-Ing. Klaus Jobmann
Tag der Promotion	23. August 2002

Vorwort

Die Grundlage der vorliegenden Arbeit ist im Rahmen eines Projekts für den DFN-Verein entstanden. Mein erster Dank gilt Frau Gerti Foest. Mit ihrer bestimmten, aber auch interessierten und fürsorglichen Betreuung im DFN-Verein hat sie in hohem Maß dazu beigetragen, dass das Projekt trotz teilweise schwieriger Umstände über einen Zeitraum von fast fünf Jahren erfolgreich durchgeführt werden konnte.

Weiterhin gilt mein besonderer Dank meinem ehemaligen Kollegen Jens-Sönke Vöckler. Sein kenntnisreicher Umgang mit Solaris und Squid sowie sein steter Wille zur verantwortungsbewussten Optimierung führten dazu, dass wir die DFN-Cache-Server dauerhaft und – was keine Selbstverständlichkeit ist – zuverlässig an ihrer Leistungsgrenze betreiben konnten. Ohne seine Mitarbeit wäre die Durchführung des Projekts über die genannte Dauer nicht möglich gewesen.

Herrn Prof. Pralle danke ich nicht nur für die Gelegenheit, dass ich das Projekt und weitere Aufgaben an seinem Lehrgebiet übernehmen durfte. Vielmehr danke ich ihm für sein Vertrauen und die Selbstständigkeit, die er mir – nicht ohne seine gezielte Hilfe und Anweisung – in meiner Arbeit gewährt hat.

Herrn Prof. Jobmann danke ich für die Übernahme des Koreferats sowie für seine hilfreichen Anmerkungen zu der Arbeit.

Astrid Tessmer hat als fachfremde Korrekturleserin nicht zuletzt einen Beitrag zur Verbreitung der neuen deutschen Rechtschreibung geleistet. Auch für ihre kritischen Anmerkungen zu Formulierungen und zur Gestaltung der Arbeit danke ich sehr.

Dass die Anspannung während der Ausarbeitung einer Dissertation auch in den privaten Bereich getragen wird, habe ich erwartet. Wie notwendig dabei Rückhalt und Unterstützung sind, habe ich erfahren dürfen. Meiner Familie gebührt daher mein letzter und größter Dank.

Hannover, den 2. September 2002

Christian Grimm

Kurzfassung

Mit der vorliegenden Arbeit wird untersucht, welche Kommunikationsbeziehungen und Verkehrsflüsse bei der Zusammenschaltung dedizierter Cache-Server im World Wide Web entstehen und welche Erkenntnisse sich daraus für die Planung komplexer Cache-Verbünde ergeben.

Caching ist eine in der Datenverarbeitung bewährte Methode zur Schonung von Ressourcen und zur Reduzierung von Zugriffszeiten. Neben den bekannten Cache-Speichern auf Mainboards oder Festplatten lässt sich das Prinzip des Caching auch auf komplexere und räumlich verteilte Systeme wie das World Wide Web übertragen. Hier nehmen dedizierte Cache-Server die Anfragen von WWW-Klienten entgegen und leiten sie an die entsprechenden WWW-Server weiter. Die Antworten der WWW-Server werden auf den Cache-Servern gespeichert und an die WWW-Klienten weitergeleitet. Spätere Anfragen nach denselben Inhalten können direkt aus dem Speicher der Cache-Server beantwortet werden, wodurch sich das Verkehrsaufkommen mit den WWW-Servern verringert. Der Einsatz von Cache-Servern im World Wide Web bietet demnach zwei wesentliche Vorteile. Zum einen werden die Kapazitäten auf den Außenanbindungen des Netzes geschont, zum anderen reduzieren Treffer auf dem Cache-Server die Zugriffszeiten für die Nutzer.

Aus dem Zusammenschluss mehrerer Cache-Server lassen sich Cache-Verbünde bilden. Die Vorteile eines Cache-Verbundes sind offensichtlich. Sofern sichergestellt ist, dass Objekte nur einmal innerhalb des gesamten Cache-Verbundes gespeichert werden, steht als gesamter Speicher die Summe der Speicher aller Cache-Server zur Verfügung. Zudem bietet ein Cache-Verbund im Vergleich zu einzelnen Cache-Servern durch gezielte Lastverteilung und Bildung von Redundanzen einen wesentlich höheren Durchsatz sowie eine bessere Ausfallsicherheit.

Die Bildung von Cache-Verbünden stellt neue Anforderungen an die Implementierung von Cache-Servern. Da sich die Kommunikation innerhalb eines Cache-Verbundes ausschließlich über HTTP als ineffizient erwiesen hat, werden heute eigenständige Protokolle für die Inter-Cache-Kommunikation eingesetzt. Zu diesen Protokollen zählen unter anderem das Internet Cache Protocol (ICP), das Hypertext Caching Protocol (HTCP) sowie die Cache Digests.

Im Rahmen eines durch den Verein zur Förderung eines deutschen Forschungsnetzes e. V. (DFN-Verein) mit Mitteln des BMBF geförderten Projektes wurde vom RVS ein zentraler Cache-Verbund im deutschen Wissenschaftsnetz aufgebaut und betrieben. Über einen Zeitraum von vier Jahren zählte dieser DFN-Cache-Verbund zu den integralen Diensten im deutschen Wissenschaftsnetz. Der DFN-Cache-Verbund wurde zeitweise von über 200 lokalen Cache-Servern an Hochschulen und Forschungseinrichtungen genutzt.

Im ersten Teil der vorliegenden Arbeit werden die Verkehrsstatistiken des DFN-Cache-Verbundes zu ausgewählten Zeitabschnitten eingehend analysiert. Neben der Herleitung allgemeiner Aussagen über die Entwicklung des World Wide Web in den vergangenen Jahren liegt der Schwerpunkt der Analysen auf einer quantitativen Betrachtung des gesamten Verkehrsaufkommens und auf der Formulierung allgemeiner Kommunikationsbeziehungen in einem Cache-Verbund. Hierbei werden sowohl die eigentlichen Nutzdaten über HTTP als auch die verwendeten Protokolle zur Inter-Cache-Kommunikation untersucht. Als Ergebnis werden die Parameter der charakteristischen Verkehrslasten aus den betrachteten Zeiträumen definiert und durch mathematische Modelle approximiert, ferner werden mit den gewonnenen Daten die formulierten Kommunikationsbeziehungen verifiziert. Da innerhalb des betrachteten Zeitraumes sowohl die auf den DFN-Cache-Servern eingesetzte Cache-Software als auch die zur Verfügung stehende Hardware optimiert wurde, lassen sich durch weiterführende Überlegungen auch Anforderungsprofile für hochleistungsfähige Cache-Verbünde ableiten.

Die aus den Analysen gewonnenen Daten werden im zweiten Teil der Arbeit in ein Modell eingebracht, dessen Ziel die Simulation der Verkehrsflüsse in einem Cache-Verbund ist. Das Modell bietet eine grafische Oberfläche auf Basis der Qt-Bibliothek und wurde im Rahmen der Arbeit entwickelt. Die Anzahl der Cache-Server sowie die eingesetzte Inter-Cache-Kommunikation kann für jeden simulierten Cache-Verbund frei gewählt werden. Als Eingangslast des Simulationsmodells werden die ermittelten charakteristischen Verkehrslasten verwendet, die sich durch Manipulation einzelner Parameter gezielt variieren lassen. Die kennzeichnenden Eigenschaften der verschiedenen Inter-Cache-Protokolle werden bei der Simulation berücksichtigt. Für die Ermittlung ausfalltoleranter Konfigurationen lässt das Modell die gezielte Abschaltung einzelner Cache-Server zu. Mit dem Simulationsmodell steht ein effizientes Werkzeug zur Verfügung, das die Bewertung von Cache-Verbünden hinsichtlich Verkehrsverteilung, Lastverhalten und Ausfalltoleranz ermöglicht.

Schlagwörter: WWW-Caching, Cache-Verbund, Inter-Cache-Kommunikation.

Abstract

This thesis paper investigates which communications patterns and data traffic flows are created by interconnecting dedicated cache servers in the World Wide Web and the results thereof on planning complex cache hierarchies.

In data processing, caching is a proven method of protecting resources and reducing access times. In addition to the conventional cache memory on mainboards and hard disk drives the principle of caching can also be applied to more complex distributed systems such as the World Wide Web. Here dedicated cache servers accept the requests of web clients and forward them to the respective web servers. The web servers' responses are stored on cache servers and then forwarded to the web clients. Following requests regarding the same content can be replied to directly from the cache servers' memory, reducing traffic with web servers. Thus, the use of cache servers in the World Wide Web offers two significant advantages. First, resources on inbound links are protected and secondly, hits on the cache server reduce access times for the users.

By interconnecting multiple cache servers, cache hierarchies can be created. The advantages of a cache hierarchy are obvious. Provided that objects are only stored once within the entire cache hierarchy the sum of individual cache servers is available for memory space. Furthermore, a cache hierarchy creates a higher throughput and a higher fault tolerance in comparison to single cache servers by offering a targeted load sharing and the creation of redundancies.

However, the creation of cache hierarchies does set a number of new requirements for the implementation of cache servers. Since the communication within a cache hierarchy exclusively via HTTP has proven to be inefficient, self-contained protocols for the inter cache communication are applied today. Internet Cache Protocol (ICP), Hyper Text Caching Protocol (HTCP), Cache Array Protocol (CARP) and Cache Digests are some of these protocols.

Within a project of the German Research Network Association (DFN-Verein) sponsored with funds of the BMBF a central cache hierarchy was erected and operated by the RVS in the German Research Network. Over a period of four years this DFN cache hierarchy was an integral service within the German Research Network. At

times, the DFN cache hierarchy was used by more than 200 local cache servers in universities and research institutions.

The first part of this paper analyses in-depth traffic statistics in the DFN cache hierarchy during selected periods. In addition to general reflections on the development of the World Wide Web in the past years, the analysis focuses on a quantitative reflection of the total traffic volume. Both the original user data via HTTP as well as the data of inter cache communication are investigated. As a result of these analysis characteristic workloads of web traffic over the past four years are defined. In a next step approximations of the respective distribution functions lead to mathematical models. Additionally the correlation between user data and the emerging inter cache communication is defined and verified by the measured data. Since both the cache software used on the DFN cache servers as well as the available hardware were optimized during the time of research the results can be also used to derive profiles for high performance caching hierarchies.

A load generator based on the data derived from the analyses leads to a simulation model for cache hierarchies, which represents the second part of this research. Based on the Qt library the model contains an interactive graphical interface to monitor and controls the progress of the simulation. The underlying implementation of a cache hierarchy considers different numbers of cache servers and cache partitions as well as different inter cache protocols. The simulation model provides an efficient tool that allows for the evaluation of cache hierarchies in regard to traffic distribution, load sharing and fault tolerance.

Key words: web caching, cache hierarchy, inter cache communication.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Abkürzungen

BMBF	Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie
B-WiN	Breitband-Wissenschaftsnetz des DFN
CARP	Cache Array Routing Protocol
CCDF	Complementary Cumulative Distribution Function
CDF	Cumulative Distribution Function
CDN	Content Distribution Network
CERNET	Chinese Education And Research Network
CGI	Common Gateway Interface
CGMP	Cache Group Management Protocol
DE-CIX	Deutscher Commercial Internet eXchange
DFN	Verein zur Förderung eines Deutsches Forschungsnetzes e. V.
DNS	Domain Name Service
ESNET	Energy Sciences Network
FTP	File Transfer Protocol
GMT	Greenwich Mean Time
G-WiN	Gigabit Wissenschaftsnetz des DFN
HMAC	Keyed-Hash Massage Authentication Code
HTCP	Hypertext Caching Protocol
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol

HTTP-NG	Hypertext Transfer Protocol Next Generation
HvStd	Hauptverkehrsstunde
ICC	Inter Cache Communication
ICP	Inter Cache Protocol
IETF	Internet Engineering Task Force
IMS	If-Modified-Since
IP	Internet Protocol
ISDN	Integrated Services Digital Network
ISO	International Organisation for Standardization
ITU	International Telecommunication Union
LAN	Local Area Network
LRU	Least Recently Used
MD5	Message Digest 5
MIME	Multipurpose Internet Mail Extensions
MLE	Maximum Likelihood Estimator
MSS	Maximum Segment Size
NLANR	National Laboratory for Applied Network Research
NNTP	Network News Transport Protocol
NSFNET	National Science Foundation Network
NTP	Network Time Protocol
OSI	Open Systems Interconnection
P2P-Netz	Peer-to-Peer-Netz
PDF	Probability Density Function
QoS	Quality of Service
RAID	Redundant Array of Inexpensive Disks
RAM	Random Access Memory
RFC	Request for Comments

RTSP	Real Time Streaming Protocol
RTT	Round Trip Time
RUP	Resource Update Protocol
RRZN	Regionales Rechenzentrum für Niedersachsen
RVS	Lehrgebiet Rechnernetze und Verteilte Systeme
SGML	Standard Generalized Markup Language
SMIL	Synchronized Multimedia Integration Language
SMTP	Simple Mail Transfer Protocol
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TEN	Trans European Network
TLD	Toplevel Domain
TTL	Time To Live
UDP	User Datagram Protocol
UFS	Standard Unix File System
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
W3C	World Wide Web Consortium
WAIS	Wide Area Information Service
WAN	Wide Area Network
WCCP	Web Cache Coordination Protocol
WEBI	Web Intermediaries (IETF working Group)
WREC	Web Replication and Caching (IETF working Group)
WWW	World Wide Web
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language

Kapitel 1

Einleitung

1.1 Einführung und Motivation

Kein anderer Dienst hat die Entwicklung des Internet so nachhaltig beeinflusst wie das World Wide Web. Mit der Möglichkeit, Daten auf selbst gestalteten grafischen Oberflächen zu präsentieren, eigene Inhalte mit weltweit verteilten Informationen zu verknüpfen sowie beliebige Dateiformate zu transportieren, hat das World Wide Web längst seinen wissenschaftlichen Ursprung verlassen und das Internet auch im kommerziellen, öffentlichen und privaten Bereich etabliert. Auch die Feststellung, dass wir uns in einer *vernetzten Informationsgesellschaft* befinden [Cai96], wird maßgeblich durch die Technologie des World Wide Web getragen.

Mit der steigenden Popularität des World Wide Web und der zunehmenden Belastung der Netze durch WWW-Verkehr ging jedoch bald Unzufriedenheit über hohe Wartezeiten einher. Überlastete Server und fehlende Übertragungskapazitäten auf den Leitungen waren die Ursachen dieser Situation. Als Abhilfe wurde, analog zu der in der Datenverarbeitung bereits etablierten Technik des Caching, der Einsatz von WWW-Cache-Servern vorgeschlagen.

WWW-Cache-Server speichern abgerufene Inhalte aus dem World Wide Web lokal in der Nähe der Nutzer und stellen die Inhalte bei erneutem Abruf unmittelbar zur Verfügung. Die Vorteile des Caching im World Wide Web kommen gleichzeitig verschiedenen an den Kommunikationsabläufen beteiligten Gruppen zugute:

- Nutzer: Objekte aus dem Cache-Server können in der Regel mit deutlich höherer Datenrate übertragen werden als vom originalen WWW-Server. Dadurch werden Wartezeiten reduziert, so dass die Nutzbarkeit des World Wide Web steigt. Weiterhin verringern sich durch kürzere Sitzungsdauern die anfallenden zeitabhängigen Verbindungsentgelte.
- Netzbetreiber: Der WWW-Verkehr auf den Leitungen im Kernnetz und an den Übergangspunkten in andere Netze wird verringert.
- Betreiber von WWW-Servern: Die Anzahl eingehender Requests, die von den Servern verarbeitet werden müssen, sinkt. Entsprechend reduziert sich auch das

Datenvolumen des ausgehenden WWW-Verkehrs. Die Last auf den WWW-Servern sinkt.

Um den Nutzen von Caching im World Wide Web zu steigern, wurde in einem nächsten Schritt die Bildung von Verbünden aus Cache-Servern vorgeschlagen. Es zeigte sich jedoch, dass das dem World Wide Web zugrunde liegende *Hypertext Transfer Protokoll* (HTTP) für die alleinige Kommunikation innerhalb eines Cache-Verbundes nicht geeignet ist. Erst mit der Einführung spezieller Inter-Cache-Protokolle, wie zuerst dem *Internet Cache Protocol* (ICP) und später unter anderem den *Cache Digests* oder dem *Cache Array Routing Protocol* (CARP), konnte die Bildung effizienter Cache-Verbünde realisiert werden.

Eine notwendige Voraussetzung für die Bewertung der Effizienz und den Betrieb eines Cache-Verbundes ist die Analyse des anfallenden Inter-Cache-Verkehrs. Diese Analyse gestaltet sich jedoch äußerst komplex, da die Kommunikationsabläufe sowie das übertragene Datenvolumen innerhalb eines Cache-Verbundes von mehreren Faktoren anhängen:

- von den eingesetzten Inter-Cache-Protokollen,
- von dem Grad der Vermaschung der Caches,
- von der Anzahl der Hierarchiestufen im Cache-Verbund,
- von den gewählten Verfahren zur Verkehrslenkung innerhalb des Verbundes.

Entsprechende Untersuchungen von Inter-Cache-Protokollen und Kommunikationsbeziehungen in Cache-Verbünden werden sogar als eine der *Grand Challenges* der Modellierung von Verkehr im Internet angesehen [CLR00]. Eine zusätzliche Schwierigkeit derartiger Untersuchungen liegt darin, dass notwendige Rohdaten aus Cache-Verbünden kaum für wissenschaftliche Untersuchungen zur Verfügung stehen.

Als Folge der mangelnden grundlegenden Betrachtungen fehlen heute Verfahren und Modelle, um größere Cache-Verbünde zuverlässig zu konzipieren. Die daraus resultierenden Fragestellungen betreffen unter anderem die Anzahl einzusetzender Cache-Server, die Wahl geeigneter Inter-Cache-Protokolle sowie das Verhalten des gesamten Verbundes bei Ausfall einzelner Cache-Server oder Leitungen. Für den Betreiber ist somit eine Abschätzung von Aufwand gegenüber Nutzen bei dem Aufbau eines Cache-Verbundes nur schwer möglich.

1.2 Ziele

Um den Aufbau eines Cache-Verbundes im Deutschen Wissenschaftsnetz voranzutreiben, wurde am Lehrgebiet Rechnernetze und Verteilte Systeme der Universität Hannover das Projekt *Konzeption einer Cache-Server-Infrastruktur im Wissenschaftsnetz* durchgeführt. Im Rahmen des Projekts konnte während eines Zeitraumes von über vier Jahren ein Cache-Verbund aus mehreren übergeordneten DFN-Caches

sowie bis zu 200 lokalen Caches von Hochschulen oder Forschungseinrichtungen aufgebaut und betrieben werden[1].

Die Beobachtungen, die mit dem DFN-Cache-Verbund möglich waren, lieferten zunächst wesentliche Einblicke in das Wachstum und die Veränderungen im World Wide Web. Für den ersten Teil der vorliegenden Arbeit wurden fünf Zeiträume von jeweils vier Wochen ausgewählt, in denen die gesammelten Verkehrsdaten eingehend analysiert wurden. In den Analysen werden charakteristische Merkmale des WWW-Verkehrs untersucht, wie z. B. die Verteilung der Objektgrößen, Mimetypes, HTTP-Codes, Übertragungsraten, Toplevel Domains der WWW-Server oder die Zwischenankunftszeiten der Anfragen. Aus einem Vergleich der Ergebnisse werden in einem weiteren Schritt Aussagen über die mögliche zukünftige Entwicklung des WWW-Verkehrs im Wissenschaftsnetz abgeleitet.

Während der Projektlaufzeit wurden unterschiedliche Konfigurationen und Protokolle zur Inter-Cache-Kommunikation im DFN-Cache-Verbund eingesetzt. Die Auswertung der jeweiligen Verkehrsstatistiken lässt eine Bewertung dieser Protokolle zu. Zugleich können die Kommunikationsbeziehungen in Cache-Verbünden formuliert und anhand der Ergebnisse verifiziert werden. Mit den vorgestellten Ergebnissen aus dem praktischen Einsatz wird ein wichtiger Beitrag zur effizienten Konfiguration und optimalen Auswahl der Inter-Cache-Protokolle geliefert.

Im zweiten Teil der Arbeit wird ein Simulationsmodell entworfen, in das die Daten der Auswertungen eingebracht werden. Die erste Komponente des Modells besteht aus einem eigenständigen Lastgenerator, der synthetischen WWW-Verkehr erzeugt. Mit dem Generator können die charakteristischen Workloads aus den fünf untersuchten Zeiträumen sowie eine weitere, frei definierbare Workload erzeugt werden.

Mit der zweiten Komponente des Simulationsmodells wird ein Cache-Verbund nachgebildet. In diesem Modell sind die Anzahl der Cache-Server, die Konfiguration des Verbundes sowie die eingesetzten Inter-Cache-Protokolle einstellbare Variablen. Der ein- bzw. ausgehende Verkehr in dem simulierten Cache-Verbund wird durch den Lastgenerator erzeugt. Ziel des gesamten Modells ist es, die Verkehrsbeziehungen innerhalb eines Cache-Verbundes sowie dessen Verhalten unter realistischen Bedingungen hinsichtlich Lastverteilung, Ausfalltoleranz und Einsparpotential zu untersuchen. Um diesen Anforderungen gerecht zu werden, lassen sich während der Simulation einzelne Caches gezielt deaktivieren.

Das Simulationsmodell bietet eine grafische Oberfläche auf Basis der Qt-Bibliothek und wurde für UNIX-Systeme entwickelt. Mit dem vorgestellten Ergebnis liegt ein Werkzeug vor, das unterstützend bei der Konzeption von Cache-Verbünden eingesetzt werden kann.

---

1. Das Projekt wurde aus Mitteln des Bundesministeriums für Bildung, Wissenschaft, Forschung und Technologie (BMBF) durch den Verein zur Förderung eines Deutschen Forschungsnetzes e. V. (DFN-Verein) finanziert.

1.3 Aufbau der Arbeit

In Kapitel 2 werden die Grundlagen betrachtet, auf die im weiteren Verlauf der Arbeit zurückgegriffen wird. Neben der Architektur des World Wide Web und wichtigen Aspekten des Caching stehen Kommunikationsverfahren in Cache-Verbünden im Vordergrund. Mit diesem Kapitel wird auch der gegenwärtige Stand in der Literatur zu den in der vorliegenden Arbeit betrachteten Themen dargelegt.

Der erste Teil von Kapitel 3 beschreibt die Durchführung des DFN-Cache-Projektes. Neben der Darstellung der verwendeten Hard- und Software werden verschiedene Konzeptionen des DFN-Cache-Verbundes erläutert, die während des Projektzeitraumes eingesetzt wurden. Die Analyse der im DFN-Cache-Verbund gesammelten Verkehrsstatistiken und die Formulierung allgemeiner Kommunikationsbeziehungen in einem Cache-Verbund bilden den zentralen Teil des Kapitels.

Für die Implementierung des Lastgenerators werden in Kapitel 4 die in einem Cache-Verbund auftretenden Datenströme modelliert. Dabei werden zunächst die Parameter bestimmt, von denen die betrachteten Merkmale Objektgröße, Datenrate und Zwischenankunftszeit abhängen. In einem weiteren Schritt wird ein Verfahren vorgestellt, mit dem sich die für die Modellierung relevanten Datenströme identifizieren lassen. Der Schwerpunkt des Kapitels liegt in der mathematischen Modellierung der Datenströme durch Approximation ausgewählter Verteilungsfunktionen.

In Kapitel 5 wird das im Rahmen der Arbeit entwickelte Simulationsmodell vorgestellt. Basierend auf den Verkehrsanalysen aus dem vorangegangenen Kapitel liegt der Schwerpunkt auf einer detaillierten Modellierung der Inter-Cache-Protokolle sowie der Kommunikationsbeziehungen in Cache-Verbünden. Neben der Beschreibung des Designs und verschiedenen Aspekten der Implementierung werden Fallbeispiele für die Einsatzmöglichkeiten des Modells aufgeführt.

Mit der Zusammenfassung der Ergebnisse in Kapitel 6 schließt die Arbeit. Im Mittelpunkt stehen hierbei grundlegende Aussagen über die Einsatzmöglichkeiten von Cache-Verbünden. Zugleich werden mögliche Erweiterungen des entwickelten Simulationsmodells dargelegt. Ein Ausblick betrachtet die zukünftige Bedeutung des Caching im Vergleich zu neuen Verfahren für die Verteilung von Inhalten im World Wide Web.

Im Anhang der Arbeit werden begleitende Untersuchungen aufgeführt, die nicht in dem DFN-Cache-Verbund durchgeführt werden konnten. Aus den Untersuchungen werden Annahmen für die Modellierung abgeleitet. Ein zweiter Teil des Anhangs fasst die mathematischen Verfahren und Verteilungsfunktionen zusammen, die bei der Verkehrsanalyse und der Modellierung angewendet wurden.

Kapitel 2

Grundlagen

2.1 Das World Wide Web

2.1.1 Entwicklung und Bedeutung

Die in den letzten Jahren zunehmende Bedeutung des World Wide Web lässt sich anhand verschiedener Statistiken quantitativ belegen. Abbildung 2.1 zeigt die Anzahl der Hosts und WWW-Server im Internet seit 1995. Sowohl die Anzahl der Hosts als auch die Zahl der verfügbaren WWW-Server stieg bis Anfang 2001 mit annähernd exponentiellem Verlauf stetig an.

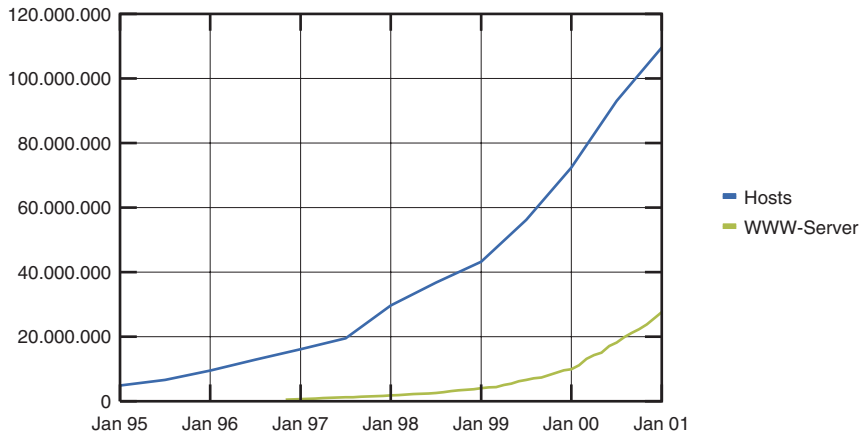


Abbildung 2.1: Anzahl Hosts und WWW-Server im Internet seit 1995

Es ist zu beachten, dass nicht jeder WWW-Server einem einzelnen Host entspricht, da mehrere WWW-Server mit unterschiedlichen Namen auf einem physikalischen System gleichzeitig betrieben werden können. Dennoch steht jeder WWW-Server in dieser Statistik für ein eigenständiges Informationsangebot. Daher wird mit dieser Statistik belegt, dass die Zahl der Informationsangebote im World Wide Web steigt und es gleichzeitig immer mehr Rechner im Internet gibt, von denen aus auf diese Angebote zugegriffen werden kann.

Über das verfügbare Datenvolumen im World Wide Web gibt es keine zuverlässigen Angaben. Anfang 1999 wurde der Umfang von ca. 800 Millionen textbasierten Seiten auf 15 TByte geschätzt [LaGi99]. Eine entsprechende Hochrechnung mit Angaben von Suchmaschinen, die aktuell ca. 1,6 Milliarden Seiten indiziert haben [Goo01], lässt heute auf einen Umfang der textbasierten Inhalte von 30 TByte schließen. Zusammen mit den nicht indexierten Seiten sowie Multimedia- und Applikationsdateien, deren Anteil und Größe die der textbasierten Dateien deutlich übertrifft (s. Kapitel 3.2.3), lässt sich das gesamte Volumen des World Wide Web derzeit auf über 100 TByte abschätzen. Vergleichbare Schätzungen bestätigen diese Größenordnung [ACGM+01].

Für die Planung und den Betrieb von Netzen sind Aussagen über die theoretisch verfügbaren Inhalte nur von geringer Bedeutung. Entscheidend ist das tatsächlich übertragene Datenvolumen. In Abbildung 2.2 wird der IP-Verkehr am Übergabepunkt von MCI Worldcom in das deutsche Wissenschaftsnetz nach Anteilen verschiedener Dienste aufgeschlüsselt[2]. Der gewählte Messpunkt ist aus betrieblicher Sicht von besonderem Interesse, da die ermittelten Werte dem Verkehr auf der Transatlantikstrecke zwischen Washington und Hamburg entsprechen.

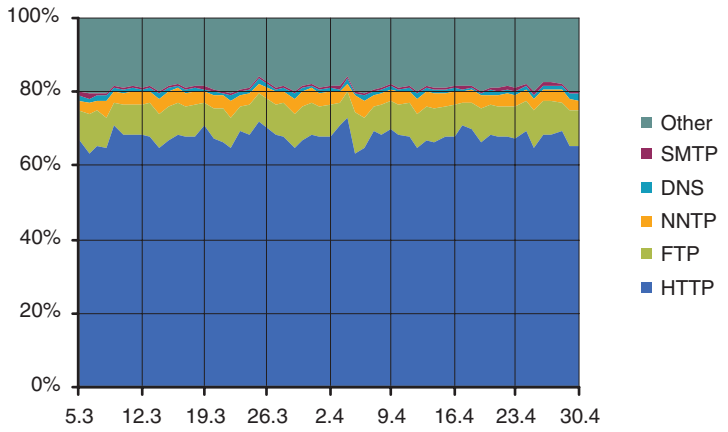


Abbildung 2.2: Verkehr MCI → Wissenschaftsnetz, 5. März – 30. April 1998

Der Anteil des WWW-Verkehrs überwiegt mit ungefähr 65% im Tagesmittel deutlich. Ähnliche Anteile von annähernd 70% werden in Untersuchungen aus anderen Kernnetzen, so z. B. MCI Worldcom [CMT98], für denselben Zeitraum genannt.

Aus der Verteilung der übertragenen Datenvolumen darf nicht die Bedeutung des World Wide Web im eigentlichen Sinne abgeleitet werden. So haben Email oder der Domain Name Service (DNS) keinen geringeren Stellenwert für den Nutzen und die Funktion des Internet als das World Wide Web. Jedoch wird deutlich, dass in dem

2. Die Statistik wurde mit der Software Netflow von Cisco durch Auswertung von TCP-Ports aus den IP-Paketen erstellt. In den 20% sonstigem Verkehr sind noch WWW-Dienste enthalten, die nicht den regulären TCP-Port 80 für HTTP nutzen.

WWW-Verkehr eine wesentliche Ursache für die Belastung der Netze im Internet zu sehen ist.

Bei der Planung von IP-Netzen muss daher der WWW-Verkehr und dessen zukünftige Entwicklung maßgeblich berücksichtigt werden. Besonders wichtig sind diese Überlegungen für die Auslegung der Kernnetze sowie der Übergangspunkte (Peering Points) in andere Netze. Weiterhin kann angenommen werden, dass die Verringerung des WWW-Verkehrs ein wirksames Mittel darstellt, um die Lastsituation in überfüllten Netzen zu verbessern.

2.1.2 Architektur

2.1.2.1 Semantische Komponenten

Das World Wide Web wurde als ein eigenständiges System zur Übertragung und Darstellung von hypertext-basierten Informationen über das Internet konzipiert. Bei der Umsetzung mussten folgende grundlegenden Anforderungen erfüllt werden:

- Adressierung der Ressourcen,
- Übertragung der Daten,
- Darstellung der Informationen.

Im Folgenden werden die einzelnen Spezifikationen dieser drei semantischen Komponenten des World Wide Web [KrRe01] näher betrachtet. Die Festlegung der Spezifikationen wurde zuerst unter der Federführung von Tim Berners-Lee vorgenommen. Seit Oktober 1994 liegt die Verantwortung für Standardisierungen im World Wide Web in der Hand des *World Wide Web Consortium* (W3C), dessen Direktor Berners-Lee heute ist.

Adressierung der Ressourcen

Die generische Syntax zur Identifizierung abstrakter oder physischer Ressourcen im Internet wird über *Uniform Resource Identifier* (URI) festgelegt [BLFM98]. Innerhalb dieser Spezifikation werden *Uniform Resource Locator* (URL) definiert, die somit eine Untermenge der URI bilden. Mit der Beschreibung von URI und URL innerhalb eines gemeinsamen Dokumentes wurden Unklarheiten ausgeräumt, die aus vergangenen, getrennt formulierten Definitionen herrührten [BL94][BLMM94] [Fie95].

URLs stellen die gebräuchliche Form der Adressierung von Ressourcen im World Wide Web dar. Eine URL besteht aus drei Elementen:

- Protokoll: Bezeichnung des auf der Anwendungsebene verwendeten Protokolls. Neben `http` können auch andere Dienste, wie z. B. `ftp`, `gopher`, `telnet`, `news` oder `rtsp`, verwendet werden.
- Adresse: symbolischer Name oder IP-Adresse des WWW-Servers. Zusätzlich

muss der TCP-Port angegeben werden, falls dieser vom Standard-Port des verwendeten Anwendungsprotokolls abweicht.

- Dateiname: Pfad und Name der Datei relativ zum Wurzelverzeichnis des WWW-Servers.

In Abbildung 2.3 ist die formale Syntax von URLs sowie der typische Aufbau anhand eines Beispiels dargestellt.

```
<protocol>://<address>[:<port>]/<filename>

http://www.rvs.uni-hannover.de/projects/index.html
```

Abbildung 2.3: Aufbau Uniform Resource Locator

Eine weitere Form der Adressierung stellen die *Uniform Resource Names* (URN) dar [SoMa94]. Durch URNs sollen zwei Probleme behoben werden, die bei der Verwendung von URLs entstehen. URNs sind ebenfalls eine Untermenge der URIs und ermöglichen eine ortsunabhängige Adressierung. Dadurch können zum einen Objekte zwischen WWW-Servern verschoben werden, ohne dass die Adresse angepasst werden muss. Zum anderen wird ermöglicht, dass für identische Objekte auf verschiedenen WWW-Servern nur eine einzige Adresse existiert. Bei der Auflösung eines URN sollte der WWW-Server adressiert werden, der aus netztechnischer Sicht am günstigsten zu erreichen ist [DaMe97]. Die notwendigen Informationen zur Auswahl des geeigneten Servers müssen allerdings in eigenständigen Datenbanken vorgehalten werden.

Bisher sind URNs im World Wide Web ohne praktische Bedeutung, da kaum Implementierungen zur Unterstützung vorliegen. URNs werden im weiteren Verlauf dieser Arbeit nicht weiter berücksichtigt.

Übertragung der Daten

Zur Übertragung von Daten im World Wide Web wird das *Hypertext Transfer Protocol* (HTTP) eingesetzt. Die erste veröffentlichte Spezifikation des Protokolls unter der nachträglich festgelegten Versionsnummer HTTP/0.9 war äußerst einfach gehalten [BL92]. In der Beschreibung wurde lediglich festgelegt, dass HTTP ein *Request-Response*-orientiertes Protokoll zwischen WWW-Klient und WWW-Server ist. Hierbei enthält ein HTTP-Request den URL des gewünschten Objekts im Klartext. Mit der HTTP-Response wird das Objekt ohne Angabe weiterer Informationen übertragen.

Die nächste Protokollversion HTTP/1.0 [BLFF96] wurde in Anlehnung an den MIME-Standard [FrBo96a][FrBo96b] entwickelt. Eine wesentliche Erweiterung betrifft die Verwendung von Metadaten, mit denen Informationen über die angeforderten oder übertragenen Objekte (z. B. Mimetype, Datum der letzten Änderung) angegeben werden können. Zur Trennung von Metadaten und Nutzdaten werden

HTTP-Replies in *Header* und *Body* unterteilt. Das Format der Header folgt dem Standard im Internet für Textnachrichten [Cro82].

Durch die intensive Nutzung von HTTP/1.0 im World Wide Web zeigte sich jedoch bald, dass auch diese Protokollversion verbessert werden musste. Ein schwerwiegender Mangel war die geringe Performance des Protokolls, die durch das starre *Request-Response*-Verfahren verursacht wurde. Weiterhin war eine Reihe von Metadaten unklar oder unvollständig beschrieben worden, wodurch besonders die Implementierung und Nutzung von Caching im World Wide Web beeinträchtigt worden ist.

Diese Mängel wurden mit der aktuellen Version HTTP/1.1 [FGM+99] behoben. Dabei hat HTTP/1.1 gegenüber der einfachen Version HTTP/0.9 einen hohen Grad an Komplexität erreicht. Ein Indiz hierfür ist allein der Umfang der Spezifikationen, der von knapp fünf Seiten bei HTTP/0.9 über 60 Seiten bei HTTP/1.0 auf 178 Seiten angewachsen ist. Mit HTTP/1.1 ist aus Sicht des W3C vorerst das Ende der Entwicklung von HTTP erreicht:

„Now that both HTTP extensions and HTTP/1.1 are stable specifications, W3C has closed the HTTP Activity. The Activity has achieved its goals of creating a successful standard that addresses the weaknesses of earlier HTTP versions.“ [W3C01]

Ansätze für neuere Protokollversionen wie HTTP-NG (Next Generation) [LaGe99] existieren zwar, entsprechende Weiterentwicklungen werden jedoch nicht durch die *Architecture Domain* des W3C unterstützt.

Da die Eigenschaften von HTTP entscheidend für die Performance des World Wide Web sind und darüber hinaus wesentliche Bedeutung für das Caching besitzen, werden die wichtigsten Merkmale von HTTP in Kapitel 2.1.3 näher erläutert.

Darstellung der Informationen

Die Elemente zur Darstellung der Inhalte im World Wide Web wurden bisher durch die *HyperText Markup Language* (HTML) [RLHJ99] festgelegt. HTML orientiert sich im Aufbau und in der Syntax an der *Standard Generalized Markup Language* (SGML) [ISO79]. Die strukturellen Elemente, die HTML zur Verfügung stellt, sind:

- Angaben zur Formatierung und zum Aufbau der Seiten, wie z. B. Schriftgestaltung, Tabellen oder Frames,
- interaktive Elemente, wie z. B. Eingabefelder oder Auswahllisten,
- Referenzierung anderer Informationsquellen durch Links,
- Einbettung von Grafiken, Applets oder weiteren Multimedia-Objekten, die durch Erweiterungen (Plug-Ins) des WWW-Browsers innerhalb einer Seite dargestellt werden.

Das Einbringen neuer Elemente in HTML war eng an die Implementierung von WWW-Browsern gekoppelt und konnte nicht durch das W3C gesteuert werden. So existiert mittlerweile eine Reihe von HTML-Tags, die nur von bestimmten WWW-Browsern unterstützt werden. Eine weitere Schwierigkeit ist die unzureichende Konformität zu SGML. Dieser Mangel kommt besonders bei der Konvertierung anderer Datenformate von und nach HTML zum Tragen.

Die Entwicklung von HTML endet mit der derzeit aktuellen Version 4.01. Die nächste Generation, die mit der *Extensible HyperText Markup Language* (XHTML) [Pem+00] bereits vorliegt, basiert auf der *Extensible Markup Language* (XML) [BP+98]. Ziel dieser ebenfalls durch das W3C spezifizierten Sprache ist die Konformität zu SGML und damit verbunden eine bessere Standardisierung und Internationalisierung. Zur Darstellung und Steuerung multimedialer Inhalte stehen mittlerweile weitere Sprachen, wie die *Synchronized Multimedia Integration Language* (SMIL) [Aya+01], zur Verfügung.

2.1.2.2 Software-Komponenten

Wie fast alle Dienste im Internet basiert auch das World Wide Web auf der Client-Server-Architektur. In Abbildung 2.4 sind die Komponenten dieser Architektur, erweitert durch einen Proxy oder Cache, dargestellt. Die Kommunikation zwischen den Komponenten erfolgt über ein festgelegtes Protokoll, im Fall des World Wide Web über HTTP. Die Bedeutung der einzelnen Komponenten wird im Folgenden kurz erläutert.

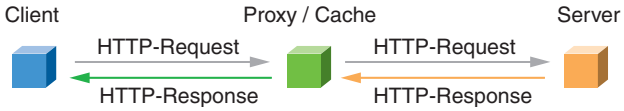


Abbildung 2.4: Client-Server-Architektur

WWW-Server

WWW-Server stellen die Informationen im World Wide Web zur Verfügung. Die prinzipielle Funktionsweise eines WWW-Servers zur Bearbeitung von HTTP-Requests lässt sich in vier Schritte unterteilen:

- Empfang des HTTP-Requests,
- Auswerten der URL nach dem gewünschten Objekt,
- Lesen des Objekts aus dem Dateisystem,
- Übertragen des Objekts in einer HTTP-Response.

Entsprechend sind die grundlegenden Anforderungen an die Implementierung von WWW-Server gering. So sind voll funktionsfähige WWW-Server für UNIX-basierte Systeme verfügbar, deren C-Quellcode deutlich weniger als 400 Zeilen umfasst

[Jon01]. Die Komplexität vieler Implementierungen ergibt sich durch zusätzliche Anforderungen. Hierzu zählen Verbesserung der Performance, Mechanismen zur Authentifizierung, Anbindung an Datenbanken, Aspekte der Sicherheit oder die Implementierung eines *Common Gateway Interface* (CGI) [Gun96] zur Ausführung von Programmen auf dem WWW-Server.

WWW-Klienten

Die Darstellung der Informationen auf den WWW-Klienten erfolgt durch die so genannten WWW-Browser. Den Browsern fällt zunächst die zentrale Aufgabe zu, den HTML-Code zu interpretieren, referenzierte Inhalte von WWW-Servern abzurufen und alle Komponenten einer Seite vollständig und korrekt auf dem Bildschirm anzuzeigen. Weiterhin ermöglichen sie den Nutzern die Interaktion über Maus oder Tastatur.

Mittlerweile verfügen Browser auch über die Möglichkeit, Programmcode wie Java, JavaScript oder ActiveX auszuführen. Diese Techniken werden häufig eingesetzt, um erhöhte und verbesserte Interaktivität mit den Nutzern zu ermöglichen. Eine zusätzliche Erweiterung wird über Plug-Ins zur Verfügung gestellt, mit denen sich herstellerabhängige Dateiformate anzeigen lassen.

Die Integration von FTP, Gopher, WAIS sowie später Email und News macht aus WWW-Browsern heute universelle Programme, mit denen verschiedenste Informationsdienste im Internet genutzt werden können.

WWW-Proxies und WWW-Caches

Ein Proxy (engl. Stellvertreter) ist eine Instanz, die Zugriffe auf Objekte stellvertretend für einen oder mehrere Prozesse ausführt [Sha86]. Der Einsatz von Proxies kann aus verschiedenen Gründen notwendig sein. So verfügen Proxies häufig über besondere Zugriffsrechte, die den eigentlichen Prozessen nicht eingeräumt werden sollen. Nur durch den Übergang über den Proxy wird den Prozessen der Zugriff auf bestimmte Objekte ermöglicht. Ebenso lassen sich Proxies zur Anpassung von Nachrichten oder Funktionsaufrufen einsetzen.

Bezogen auf Client-Server-Architekturen stellt ein Proxy ein eigenständiges System dar, das zwischen Klienten und Server geschaltet ist. Der Proxy vermittelt die Nachrichten, die zwischen Klienten und Servern ausgetauscht werden. Ein typischer Anwendungsbereich von Proxies in verteilten Systemen sind die so genannten *Application Level Gateways*, die in Verbindung mit Firewalls eingesetzt werden.

Aus technischer Sicht ist ein Cache ein Proxy, der um einen Datenspeicher erweitert wurde. In diesem Speicher werden Objekte, die vom Server ausgeliefert werden, abgelegt und können bei erneutem Abruf unmittelbar geliefert werden. Die Eigenschaften von Caches im World Wide Web werden in einem eigenständigen Kapitel näher dargelegt.

2.1.2.3 Struktur

Alle verfügbaren Informationen im World Wide Web lassen sich über URLs direkt adressieren. Dadurch konnte mit der Zeit ein dichtes Netz aus weltweit verteilten Informationsquellen gewebt werden. Dieses Netz weist keine Hierarchien und somit keine Struktur im eigentlichen Sinne auf. Die Topologie des World Wide Web lässt sich lediglich als ein teilvermaschtes Netz beschreiben, dessen Vermaschung sich mit hoher Dynamik ändert [CDK+99].

Über die Dichte der Links im World Wide Web existieren unterschiedliche Aussagen. In [BAJ00] wird angegeben, dass zwei beliebig gewählte Objekte im Mittel über 19 Links miteinander verbunden sind. Demgegenüber wird in [BKM+00] festgestellt, dass eine Verbindung zwischen zwei beliebigen Objekten überhaupt nur in 24% der Fälle existiert und diese Verbindung im Mittel 28 Links umfasst. Beide Untersuchungen stützen sich auf Statistiken, die bei Durchläufen von Suchmaschinen durch das World Wide Web, so genannten *web search crawls*, generiert wurden.

2.1.3 Das Hypertext Transfer Protocol

Mit dem Hypertext Transfer Protocol wurde ein eigenständiges Protokoll zur Übertragung von Daten im World Wide Web eingeführt. Die Entwicklung von HTTP vollzog sich in mehreren Schritten, wobei Verbesserungen im Protokollablauf, im Verbindungs-Management und in ergänzenden Header-Informationen vorgenommen wurden. In den folgenden Abschnitten werden die Eigenschaften von HTTP/1.0 und HTTP/1.1 betrachtet, die für das Caching und die Performance von HTTP von Bedeutung sind.

2.1.3.1 Grundlegende Eigenschaften

Für die gesicherte Übertragung der Objekte setzt HTTP ein verbindungsorientiertes Transportprotokoll voraus. Auch wenn es nicht explizit gefordert wird, setzen Implementierungen von HTTP in der Regel auf dem TCP/IP-Protokollstack auf.

HTTP ist nicht statusbehaftet. Das bedeutet, dass jede Folge von HTTP-Request und HTTP-Response unabhängig von vorangegangenen Übertragungen ist. Auf keiner Instanz in der WWW-Architektur werden Statusinformationen bezüglich des HTTP-Protokolls gespeichert. Um trotzdem der Forderung nach Statusinformationen nachzukommen, wurden die so genannten *Cookies* eingeführt [KrMo00]. Cookies sind nicht Bestandteil einer Spezifikation von HTTP, sondern lediglich eine proprietäre Erweiterung. Da der Einsatz von Cookies mittlerweile verbreitet ist, und Cookies den Nutzen von Caches beeinträchtigen können, werden sie in einem eigenständigen Kapitel 2.2.2.7 beschrieben.

2.1.3.2 Aufbau von Nachrichten

Der Nachrichtenaufbau von HTTP wurde bereits in Kapitel 2.1.2.1 erläutert. In Abbildung 2.5 werden die Elemente, aus denen HTTP-Request und -Response bestehen, anhand eines Beispiels dargestellt.

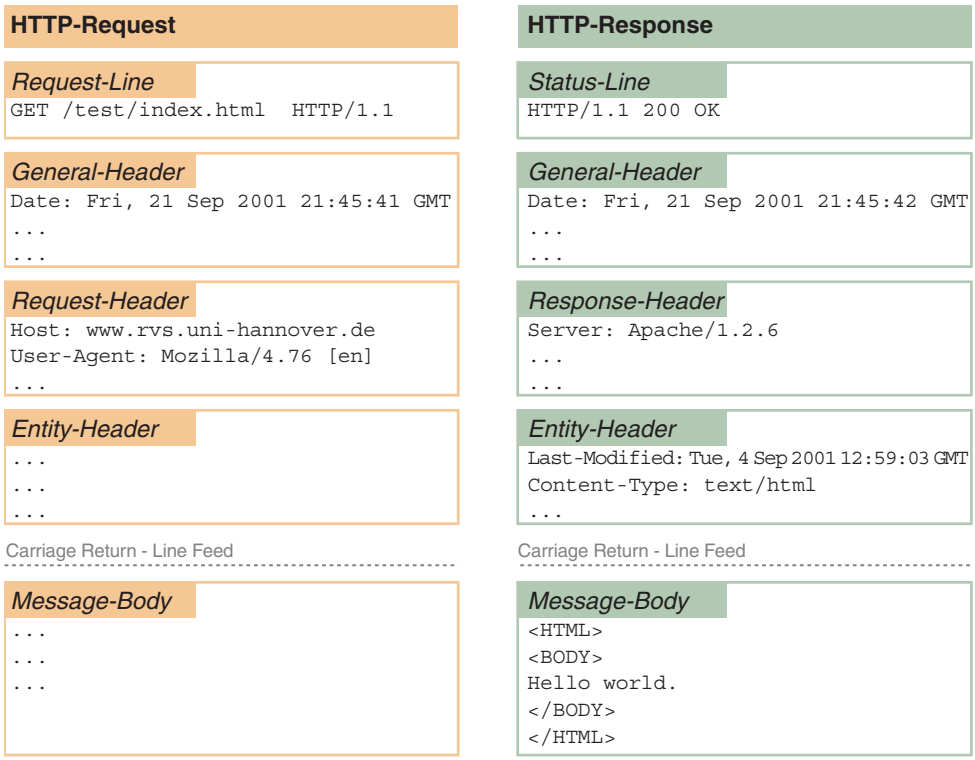


Abbildung 2.5: Aufbau von HTTP-Request und HTTP-Response

Aufbau HTTP-Request

Ein HTTP-Request wird von der *Request-Line* eingeleitet. Diese Zeile enthält die Methode des Aufrufs, den URL sowie die Versionsnummer des HTTP-Protokolls, das im WWW-Browser implementiert ist. Folgende Methoden für HTTP-Requests sind in HTTP/1.1 spezifiziert:

- GET: ruft das in dem URL angegebene Objekt vom WWW-Server ab.
- HEAD: ruft nur die Header-Informationen des in dem URL angegebenen Objekts ab.
- POST: sendet Daten an den WWW-Server. In der Regel wird diese Methode bei der Nutzung von Formularen verwendet. Die Daten werden im Body des HTTP-Requests übertragen. In dem URL ist das CGI-Programm auf dem WWW-Server angegeben, das die Daten verarbeitet.

- PUT: das in dem URL angegebene Objekt wird auf dem WWW-Server angelegt oder überschrieben. Die Daten des Objekts werden im Body des HTTP-Requests übertragen.
- DELETE: das in dem URL angegebene Objekt wird auf dem WWW-Server gelöscht.
- TRACE: ermöglicht den Test der Verbindung zum WWW-Server. Der WWW-Server kopiert alle Daten aus dem Header des HTTP-Requests in den Body der HTTP-Response. Der Nutzen dieser Methode wird bei Verwendung von Cache-Hierarchien deutlich (s. Kapitel 2.3.1).
- OPTIONS: mit dieser Methode können Informationen über den WWW-Server abgerufen werden. So lässt sich z. B. feststellen, ob die Methode TRACE von einem WWW-Server unterstützt wird.
- CONNECT: diese Methode ist zur Tunnelung von ende-zu-ende-verschlüsselter Kommunikation über Caches notwendig [Res00]. So wird ein Cache durch die CONNECT-Methode angewiesen, eine Verbindung zu dem spezifizierten WWW-Server aufzubauen und die Daten transparent über die Verbindung durchzureichen.

Es ist zu beachten, das nur die Methoden GET und HEAD für Implementierungen nach HTTP/1.1 zwingend vorgeschrieben sind. Auf weitere Details zum Aufbau von HTTP-Requests wird hier nicht näher eingegangen. Die für das Caching wichtigen Header-Informationen werden in den folgenden Kapiteln betrachtet. Für eine detaillierte Beschreibung der Header sowie einen Vergleich zwischen HTTP/1.0 und HTTP/1.1 kann neben der Spezifikation in RFC 2616 [FGM+99] auch auf [KMK99] verwiesen werden.

Aufbau HTTP-Response

Die *Status-Line* enthält die Versionsnummer des HTTP-Protokolls, das der WWW-Server zur Kommunikation mit dem WWW-Klienten verwendet, einen Statuscode, der das Ergebnis der Bearbeitung des HTTP-Requests anzeigt sowie eine Meldung im Klartext, die den Statuscode erläutern soll. Die Versionsnummer, die der WWW-Server zurücksendet, darf nicht höher sein als die, die der WWW-Klient in der Request-Line angegeben hat. Durch dieses Verfahren einigen sich WWW-Klient und WWW-Server auf die höchste gemeinsame Protokollversion. Der Klartext der Statusmeldungen wird durch RFC 2616 lediglich vorgeschlagen, Implementierungen dürfen diese Vorschläge ausdrücklich überschreiben.

Die Statuscodes werden in fünf Klassen unterteilt:

- 1xx: Informational. Reserviert für experimentelle Protokollimplementierungen, wird in der Praxis bisher nicht verwendet.
- 2xx: Success. Signalisiert die erfolgreiche Bearbeitung eines HTTP-Requests. Das geforderte Objekt ist im Body der HTTP-Response enthalten.

- 3xx: Redirection. Das geforderte Objekt kann nicht oder braucht nicht unter dem angegebenen URL ausgeliefert werden. Gegebenenfalls wird dem WWW-Klient ein alternativer URL mit der HTTP-Response übergeben.
- 4xx: Client Error. Der WWW-Server hat einen Fehler im HTTP-Request erkannt. Dieser Fehler tritt z. B. dann auf, wenn das im URL angegebene Objekt auf dem WWW-Server nicht existiert.
- 5xx: Server Error. Der WWW-Server kann wegen Überlastung oder interner Fehler (z. B. defekter Festplatte) den HTTP-Request nicht erfolgreich bearbeiten. Ein weiterer Grund für diese Meldung kann ein fehlgeschlagener Verbindungsaufbau zwischen WWW-Cache und WWW-Server sein.

Auch hier wird auf eine weiterführende Erläuterung der Header-Informationen verzichtet, indem auf die oben angegebenen Quellen verwiesen wird.

2.1.3.3 Verbindungs-Management

In Abbildung 2.6 ist der typische Protokollablauf im TCP-Stack bei der Übertragung eines Objekts im World Wide Web dargestellt.

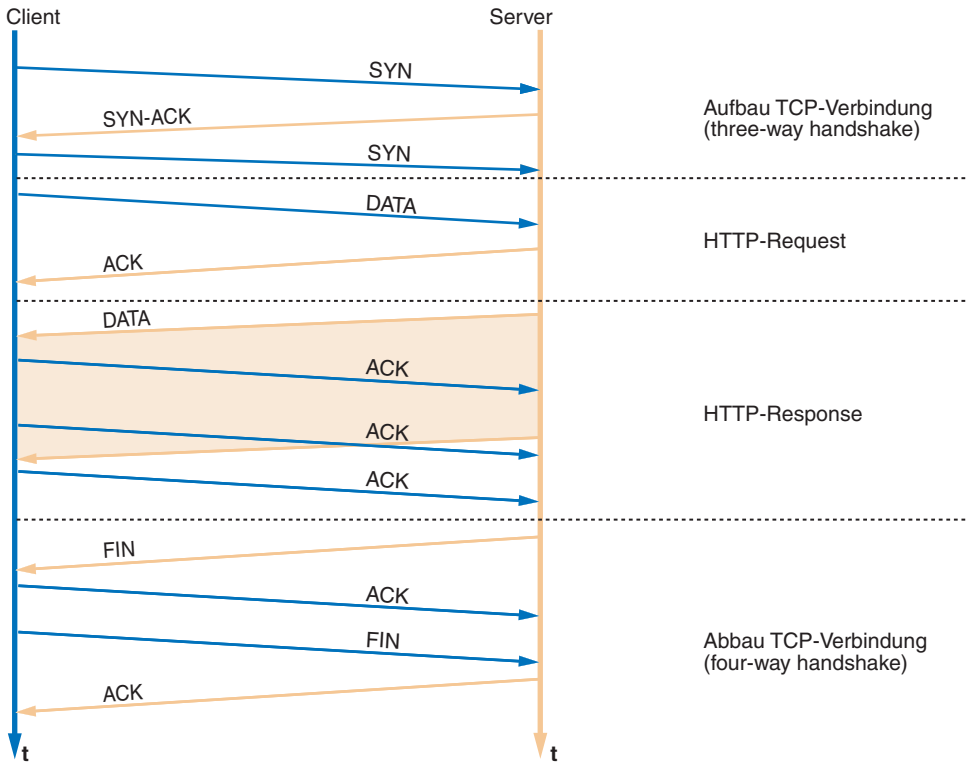


Abbildung 2.6: Übertragung eines WWW-Objekts über HTTP und TCP

Gemäß der ursprünglichen Spezifikation in HTTP/1.0 wird jedes Objekt im World Wide Web nach dem dargestellten Ablauf übertragen. Dieses Verfahren arbeitet zu-

friedenstellend, solange die zu übertragende WWW-Seite aus einem einzigen Objekt besteht. Mit der zunehmenden Entwicklung, WWW-Seiten aus mehreren Objekten wie Text und Grafik zu komponieren, stellten sich offenkundige Schwächen dieses Ansatzes dar. Im Folgenden wird als Beispiel eine WWW-Seite aus einer HTML-Datei mit zehn eingebetteten Grafiken angenommen. Anhand dieser WWW-Seite werden die Schwächen von HTTP unter Berücksichtigung der Eigenschaften von TCP erläutert.

Häufiger Verbindungsauf- und -abbau

Für die Übertragung der vollständigen WWW-Seite muss der Vorgang aus Abbildung 2.6 elf Mal nacheinander durchlaufen werden. Dabei wird zehn Mal eine Verbindung abgebaut, um sie unmittelbar danach sofort wieder aufzubauen. Setzt man als Schätzung für die Dauer von Verbindungsauf- bzw. -abbau jeweils 1 Round Trip Time (RTT) an, ergibt sich bei diesem Beispiel eine zusätzliche Wartezeit von $10 \cdot 2 \cdot 1$ RTT = 20 RTT. Bei RTTs im Weitverkehrsbereich von über 100 ms ergeben sich für komplexe Seiten allein durch den ständigen Verbindungsauf- und -abbau Wartezeiten von mehreren Sekunden.

Paketverluste

Auf stark belasteten Netzsegmenten steigt die Wahrscheinlichkeit, dass Pakete verworfen werden. So wurde 1997 in Weitverkehrsnetzen eine mittlere Verlustwahrscheinlichkeit von 5% ermittelt [Pax99]. Besonders während des Verbindungsaufbaus in TCP führen Paketverluste zu hohen Verzögerungen, da mit dem empfohlenen Timeout erst nach 3 Sekunden ein erneuter Verbindungsaufbau initiiert wird [PaAl00]. In dem genannten Beispiel besteht zehn Mal die Möglichkeit, dass durch ein verlorenes Paket während des Verbindungsaufbaus die gesamte Übertragung um mindestens 3 Sekunden verzögert wird.

Slow Start und Congestion Control

Der Sender im TCP-Protokoll sendet nicht von Beginn der Übertragung an die maximale Anzahl TCP-Pakete aus, bevor er auf eine Empfangsbestätigung wartet. In der Regel werden zuerst ein oder zwei TCP-Pakete mit der Payload entsprechend der *Maximum Segment Size* (MSS) von z. B. 522 Byte im WAN oder 1460 Byte im LAN gesendet. Nach jeder Empfangsbestätigung wird die Anzahl gesendeter Pakete gesteigert. Sobald ein Paketverlust festgestellt wird, wird die gesendete Datenmenge wieder reduziert. Dieses zyklische Vorgehen von Annähern an die maximale Datenrate und Zurücknehmen bei auftretenden Fehlern wird in dem *Slow Start* Algorithmus festgelegt [APS99]. Mit diesem Algorithmus wird verhindert, dass stark belastete oder bereits verstopfte (engl. congested) Netzsegmente durch weitere Pakete geflutet werden. Auch wenn sich der Slow Start Algorithmus in der Praxis bewährt hat, bedeutet es, dass TCP-Verbindungen nicht von Beginn an mit der maximal möglichen Übertragungsrate arbeiten. Das Maximum wird, in Abhängigkeit der MSS und der jeweiligen

Lastsituation im Netz, erst nach erfolgreicher Übertragung mehrerer Pakete erreicht. In dem genannten Beispiel wird für jedes der elf Objekte der Slow Start Algorithmus von Beginn an durchlaufen. Die effektive Datenrate, mit der die gesamte WWW-Seite übertragen wird, liegt deutlich unter der Datenrate, die bei der Übertragung eines einzelnen Objekts mit dem gleichen Datenvolumen erzielt werden kann.

Es zeigte sich, dass mit HTTP/1.0 keine zufriedenstellenden Übertragungsraten in stark belasteten Netzen möglich waren. Die Wartezeiten waren für eine intensive Nutzung des World Wide Web nicht akzeptabel. Durch die Verbreitung des World Wide Web und das zunehmende Verkehrsaufkommen verschlechterte sich diese Situation sogar. Selbst innerhalb des W3C wurde die mangelhafte Interoperabilität zwischen HTTP und TCP deutlich formuliert:

„The Internet is suffering from the effects of the HTTP/1.0 protocol, which was designed without understanding of the underlying TCP transport protocol.“ [GeNi98]

In der Literatur fanden sich bald zahlreiche Untersuchungen über das Verhalten von HTTP und TCP [Pit98]. Ein Hauptteil der Arbeiten lag in der Untersuchung der Verkehrscharakteristik von HTTP, die sich deutlich von anderen Diensten wie Telnet und rlogin oder FTP unterschied [HOT97]. So wurden anhand des HTTP-Verkehrs erstmals Untersuchungen von burst-artigem Verkehr in Weitverkehrsnetzen durchgeführt.

Einen anderen Schwerpunkt bildeten Vorschläge für ein verbessertes Verbindungs-Management, mit dem die Performance von HTTP erhöht werden sollte [NGBS+97] [PaMo95]. Diese Untersuchungen lieferten entscheidende Beiträge für neue Implementierungen und Veränderungen an der Protokollspezifikation. Im Folgenden werden drei Möglichkeiten vorgestellt, mit denen die Wartezeiten für die Nutzer oder die Belastungen auf den Netzen reduziert werden können.

Multiple Simultaneous Connections

Für dieses Verfahren sind keine Erweiterungen von HTTP/1.0 erforderlich, sondern lediglich Änderungen der WWW-Browser. Mit *Multiple Simultaneous Connections* werden die einzelnen Objekte einer WWW-Seite nicht mehr sequentiell abgerufen, sondern weitgehend parallel. Hierfür wird eine HTML-Seite bereits während der Übertragung auf eingebettete Objekte untersucht. Sobald eine Referenz auf ein eingebettetes Objekt gefunden wird, wird es über eine neue Verbindung abgerufen. Dabei wird nicht mehr auf das Ende der vorangegangenen Übertragung gewartet.

Durch diese Parallelisierung wird die Wartezeit für die Übertragung der gesamten Seite reduziert. Durch das verstärkte burst-artige Aufkommen von HTTP-Requests werden jedoch Ressourcen auf WWW-Servern, WWW-Caches oder Netzsegmenten höher belastet. Dennoch wird auch heute noch an diesem Verfahren festgehalten. Die Anzahl maximal möglicher paralleler Verbindungen ist häufig im WWW-Browser konfigurierbar, in der Regel können vier bis sechs Verbindungen gleichzeitig aufgebaut werden.

Persistent Connections

Der Ansatzpunkt für ein verbessertes und schonenderes Übertragungsverfahren lag in der Abkehr von dem ständigen Verbindungsauf- und -abbau. *Persistent Connections* wurden als Erweiterung von HTTP/1.0 vorgeschlagen und implementiert [Mog95a]. Bei diesem Verfahren werden weiterhin alle Requests sequentiell abgearbeitet, jedoch über eine einzige TCP-Verbindung. Um dieses Verfahren zu ermöglichen, mussten neue Protokollelemente in HTTP/1.0 aufgenommen werden. Folgende Anforderungen mussten für den korrekten Ablauf des Protokolls erfüllt werden:

- Der WWW-Klient muss dem WWW-Server mitteilen, dass die Verbindung nach der Übertragung eines Objekt nicht abgebrochen werden soll.
- Der WWW-Klient muss das Ende eines Objekt erkennen können – bisher wurde das Ende des Datenstroms implizit durch den Abbau der Verbindung signalisiert.
- Nach Erhalt aller Objekte muss der WWW-Klient den Verbindungsabbau initiieren.

Diese Erweiterungen lassen sich durch zusätzliche HTTP-Header realisieren. Mit dem General-Header `Connection: keep-alive` in einem HTTP-Request wird dem WWW-Server mitgeteilt, dass die Verbindung nicht abgebaut werden soll. Der WWW-Server bestätigt den Wunsch durch ein `Connection: keep-alive` im General-Header der HTTP-Response. Weiterhin fügt er im Entity-Header für jedes Objekt die Zeile `Content-Length: xxx` ein. Anhand dieser Information kann der WWW-Klient durch Abzählen der empfangenen Bytes das Ende eines Objekts erkennen. Die Persistent Connection kann vom WWW-Klient oder WWW-Server durch ein `Connection: close` aufgelöst werden.

Die Angabe der Objektgröße ist eine Voraussetzung für den korrekten Ablauf des Protokolls. Persistent Connections scheitern daher bei dynamischen Inhalten, deren Größe a priori nicht bekannt ist. In diesen Fällen muss der WWW-Server durch ein `Connection: close` die Verbindung abbauen.

Request Pipelining

Die Kombination der Vorteile von Multiple Simultaneous Connections und Persistent Connections ergibt ein Verfahren, bei dem mehrere HTTP-Requests über eine TCP-Verbindung abgearbeitet werden. Mit dem *Request Pipelining* wurde dieses Verfahren schließlich in HTTP/1.1 spezifiziert. Die Vorteile werden als so wesentlich angesehen, dass Implementierungen nach HTTP/1.1 Request Pipelining als Standard für das Verbindungs-Management verwenden sollen.

Um die Übertragung von dynamischen Inhalten auch bei Persistent Connections oder Request Pipelining zu ermöglichen, wurde weiterhin das so genannte *Chunked Encoding* eingeführt. Mit diesem Verfahren lassen sich Inhalte in einzelne Segmen-

te aufteilen und übertragen. Im Entity-Header der HTTP-Response wird mit dem Eintrag `Transfer-encoding: chunked` die Verwendung dieses Verfahrens eingeleitet. Der Body enthält zunächst eine Hexadezimalzahl, die die Länge des folgenden Datenblocks in Byte angibt. Der Datenblock folgt mit dem Beginn der nächsten Zeile. An den Datenblock schließt sich das nächste Segment aus Längenangabe und Datenblock an. Dieser Ablauf wird bis zur Übertragung des vollständigen Objekts wiederholt. Die einzelnen Blocklängen sind dabei nicht festgelegt.

2.1.3.4 Performance

Die Performance von HTTP hängt von verschiedenen Faktoren ab. Hierzu zählt in erster Linie das im vorangegangenen Kapitel dargestellte Verbindungs-Management. Auch wenn mit Implementierungen des Request Pipelining nach HTTP/1.1 eine deutliche Verbesserung erzielt werden kann [BaCr99], muss darauf hingewiesen werden, dass heute noch ein großer Teil der Übertragungen im World Wide Web über HTTP/1.0 abläuft.

Einen weiteren Faktor bilden die Eigenschaften der TCP/IP-Stacks aller an der Übertragung beteiligten Instanzen. Der TCP/IP-Stack ist Teil des Betriebssystems und wird folglich in Software implementiert. Wesentliche Eigenschaften des TCP/IP-Stacks, wie Timeouts, Fenstergrößen oder Sende- und Empfangspuffer, lassen sich durch so genannte Kernelparameter verändern. Dabei kann jedoch nur ein Kompromiss aus hoher Performance und zuverlässiger Übertragung erreicht werden. So war z. B. der TCP/IP-Stack des Betriebssystems Solaris 2.5.1 von Sun für Fileserver oder Datenbankserver in lokalen Netzen optimiert. Für den Einsatz auf Servern in Weitverkehrsnetzen mussten zunächst verschiedene Kernelparameter verändert werden, um ein zufriedenstellendes Übertragungsverhalten zu erreichen [Coc97].

Neben den genannten Verbesserungen gibt es in der Literatur weitere Ansätze, um die Übertragungszeiten im World Wide Web zu reduzieren. Dabei wurde auch die Komprimierung von Daten sowie das so genannte *Delta Encoding* untersucht. Mit Hilfe des Delta Encoding wird erreicht, dass nur noch die Abschnitte eines Objektes übertragen werden, die seit dem letzten Abruf verändert wurden [MDFK97]. Sowohl die Komprimierung als auch das Delta Encoding von Objekten wurde in HTTP/1.1 aufgenommen. Abgesehen von prototypischen Implementierungen des W3C liegen produktionsreife WWW-Server und WWW-Klienten zur Nutzung dieser Verfahren bisher nicht vor.

2.2 Caching im World Wide Web

2.2.1 Grundlagen des Caching

Um die Zugriffszeit eines Prozessors auf Informationen aus dem Speicher zu verringern, schlug M. Wilkes 1965 die Verwendung eines *Slave Memory* vor [Wil65]. Die

Aufgabe des Slave Memory war es, häufig oder zeitnah benötigte Informationen „in der Nähe“ des Prozessors vorzuhalten. Verglichen mit dem Hauptspeicher war der Slave Memory zwar wesentlich kleiner, jedoch deutlich schneller und dadurch teurer. Das Konzept des Slave Memory wurde 1968 von IBM für den Großrechner 360 Modell 85 übernommen. In der Dokumentation des Rechners wurde dieser Speicher erstmals als *Cache Memory* oder kurz *Cache* bezeichnet [Lip68].

Das Wort *caché* stammt aus dem Französischen und bedeutet wörtlich übersetzt „verborgen“ oder „versteckt“. Die Bezeichnung „Cache Memory“ entstand dadurch, dass der Cache-Speicher – im Gegensatz zum Hauptspeicher – nicht direkt von einem Programm adressiert werden kann. Die Entscheidung, welche Informationen im Cache abgelegt oder welche gelöscht werden sollen, wird allein von der Speicherverwaltung (Memory Management Unit, MMU) getroffen. Der Cache Memory ist aus Sicht der Anwendung oder des Programmierers verborgen.

Das Prinzip des Caching wird mittlerweile in fast allen Bereichen der Datenverarbeitung eingesetzt [DeGl97]. So verfügen heute z. B. Festplatten über eigenen Speicher, in dem Blocklisten oder häufig gelesene Speicherblöcke abgelegt werden. Moderne Prozessoren sind mit einem mehrstufigen On-chip-Cache ausgestattet, um Datensegmente oder Folgen von Mikrobefehlen abzulegen.

Besonders effizient lässt sich Caching in verteilten Systemen einsetzen. Verglichen mit den internen Bus-Strukturen eng gekoppelter Architekturen weist das Netzwerk, das die Knoten eines verteilten Systems miteinander verbindet, geringere Bandbreite und höhere Latenz auf. Durch Verwendung von Cache-Speichern in den einzelnen Knoten können erheblich kürzere Zugriffszeiten erzielt werden [CuSi99]. Um die Konsistenz der Daten in einem verteilten System mit mehreren Cache-Speichern aufrecht zu erhalten, ist jedoch zusätzlicher Aufwand erforderlich. In der Regel werden hierfür Cache-Kohärenz-Protokolle eingesetzt, mit denen der Ablauf für Lese- und Schreiboperationen auf Daten im Haupt- und Cache-Speicher vorgeschrieben wird [Ken87].

2.2.2 WWW-Cache-Server

Aufgrund der hohen Wartezeiten, die häufig bei der Übertragung von Daten im World Wide Web auftreten, liegt es nahe, auch hier das Prinzip des Caching anzuwenden. Zwei grundlegende Möglichkeiten für den Einsatz von Caching bieten sich an:

- Browser-Cache: bereits im WWW-Browser können Objekte im Hauptspeicher oder auf der Festplatte zwischengespeichert werden. Dieser Cache ist dann von Vorteil, wenn durch häufiges Vor- und Zurückblättern gleiche Seiten innerhalb kurzer Zeit betrachtet werden. Der Zugriff auf die Daten im Browser-Cache ist nur dem jeweiligen Nutzer gestattet.
- WWW-Cache-Server: auf eigenständigen Cache-Servern im lokalen Netz lassen sich die Inhalte, die von mehreren Nutzern abgerufen werden, zentral speichern.

Die Vorteile eines Cache-Servers kommen dann zum Tragen, wenn unterschiedliche Nutzer dieselben Inhalte abrufen.

Bereits die ersten grafischen WWW-Browser konnten Objekte im Browser-Cache auf der Festplatte zwischenspeichern [And+93]. Implementierungen von WWW-Cache-Servern waren ebenfalls früh verfügbar. So wurden auf der 1. WWW-Konferenz im Mai 1994 unabhängig voneinander drei verschiedene WWW-Cache-Server vorgestellt [Glas94] [Luo94] [Smi94].

In einer weiteren Klassifizierung lassen sich *On-demand-Cache-Server* und *On-command-Cache-Server* unterscheiden. On-demand-Cache-Server bearbeiten lediglich eingehende HTTP-Requests. Sie arbeiten ausschließlich dann, wenn tatsächlicher Bedarf eines WWW-Klienten vorliegt. Dagegen werden On-command-Cache-Server auch in folgenden Fällen selbstständig aktiv:

Validation
Der Cache-Server prüft periodisch die Inhalte im Speicher auf Aktualität. Diese Validierung erfolgt über HTTP mit den WWW-Servern (s. Kapitel 2.2.2.5) [KrWi97].

Prefetching
Der Cache-Server untersucht HTML-Seiten auf eingebettete Objekte. Wird ein solches Objekt erkannt, ruft es der Cache-Server vom WWW-Server ab. Da eine hohe Wahrscheinlichkeit besteht, dass diese Objekte auch vom WWW-Klienten abgerufen werden, dient das Prefetching einer weiteren Verringerung der Wartezeiten für die Nutzer [KLM97] [CrBa98].

Pushing
Der Cache-Server wird mit Objekten „vorgeladen“. Das Pushing bietet sich besonders dann an, wenn häufig abgerufene WWW-Seiten regelmäßig aktualisiert werden. Ein typisches Beispiel sind Online-Tageszeitungen, die nachts bei geringer Netzlast auf Cache-Server übertragen werden können. Für dieses Verfahren ist jedoch eine weitere Instanz notwendig, die die Übertragung aktualisierter Inhalte vom WWW-Server an den Cache-Server initiiert [GwSe95] [BMS96] [FJCL99].

Die genannten Ansätze für On-command-Cache-Server erscheinen zunächst vielversprechend. In der Praxis hat sich jedoch kein Verfahren etablieren können. Die Wahrscheinlichkeit, dass proaktiv übertragene oder aktualisierte Inhalte tatsächlich von Nutzern abgerufen werden, ist gering. So ist der Nutzen, der bei allen Verfahren primär in der Reduzierung der Wartezeiten für die Nutzer liegt, gegenüber dem zusätzlich generierten Verkehrsaufkommen zu gering. Darüber hinaus steigt bei allen Verfahren, besonders durch das Parsen der HTML-Seiten bei Prefetching, die Belastung der Cache-Server signifikant.

Im weiteren Verlauf der Arbeit werden ausschließlich On-demand-Cache-Server betrachtet.

2.2.2.1 Nutzen

Der ursprüngliche Grund für den Einsatz von WWW-Cache-Servern war die Verringerung der Wartezeiten bei der Übertragung von Objekten im World Wide Web. Wichtiger für die Entwicklung der Caching-Technologie im World Wide Web war jedoch das Interesse der Netzbetreiber. Ziel der Netzbetreiber war es, durch Einsatz von Cache-Servern den hohen Anteil an WWW-Verkehr im Kernnetz und an den Übergangspunkten in andere Netze zu reduzieren. Schließlich profitieren sogar die Betreiber der WWW-Server von Cache-Servern, da Cache-Server einen Teil der Requests von den WWW-Servern fernhalten.

Im Rahmen dieser Arbeit wird das Einsparpotential von WWW-Cache-Servern hinsichtlich Reduzierung der Wartezeiten und Verringerung des WWW-Verkehrs betrachtet. Anhand Abbildung 2.7 wird die Berechnung der Maßzahlen zur Bewertung von Cache-Servern dargelegt.

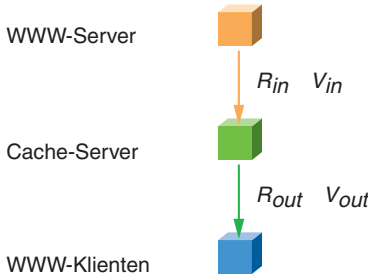


Abbildung 2.7: Berechnung der Einsparung durch Cache-Server

In der Darstellung werden die Datenflüsse an den Schnittstellen zwischen WWW-Servern und dem Cache-Server sowie zwischen dem Cache-Server und den WWW-Klienten betrachtet. R_{in} bezeichnet die Menge der HTTP-Responses, die der Cache-Server von WWW-Servern empfängt. Entsprechend bezeichnet R_{out} die Menge der HTTP-Responses, die der Cache-Server an die WWW-Klienten überträgt. Da lediglich On-demand-Cache-Server betrachtet werden, gilt $R_{in} \subseteq R_{out}$. Weiterhin kann ein kausales System vorausgesetzt werden, in dem jeder HTTP-Response ein HTTP-Request vorausgeht. Unter der Annahme, dass HTTP-Responses nicht verloren gehen können, kann an beiden betrachteten Schnittstellen die Anzahl an HTTP-Responses der Anzahl an HTTP-Requests gleichgesetzt werden.

HTTP-Responses von WWW-Servern an den Cache-Server treten nur dann auf, wenn der Cache-Server eine Anfrage nicht aus seinem Speicher beantworten kann. Die Anzahl N_{Miss} dieser so genannten Cache-Misses entspricht somit den eingehenden HTTP-Responses $|R_{in}|$. Weiterhin ergibt sich die Anzahl an Cache-Hits N_{Hit}, d. h. das Objekt befindet sich im Cache-Server, aus der Differenz von ausge-

henden zu eingehenden HTTP-Responses $|R_{out}| - |R_{in}|$ am Cache-Server. Die Objekt-Trefferrate H_O (engl. Object Hit Ratio, Request Hit Ratio) ist definiert als das Verhältnis von Cache-Hits zu insgesamt bearbeiteten HTTP-Requests:

$$H_O = \frac{N_{Hit}}{N_{Hit} + N_{Miss}} = \frac{|R_{out}| - |R_{in}|}{|R_{out}| - |R_{in}| + |R_{in}|} = \frac{|R_{out}| - |R_{in}|}{|R_{out}|} = 1 - \frac{|R_{in}|}{|R_{out}|} \quad (2.1)$$

Die Volumen-Trefferrate H_V (engl. Volume Hit Ratio, Byte Hit Ratio) ergibt sich entsprechend, wobei die Volumen bereits als skalare Größen vorausgesetzt werden:

$$H_V = \frac{V_{out} - V_{in}}{V_{out}} = 1 - \frac{V_{in}}{V_{out}} \quad (2.2)$$

Aus der Volumen-Trefferrate ergibt sich sofort der Betrag, um den der Cache-Server den eingehenden WWW-Verkehr verringert. Die Reduzierung der Wartezeiten kann dagegen nicht absolut angegeben werden, da für jeden Cache-Hit die Information fehlt, wie lange die Übertragung von dem originalen WWW-Server gedauert hätte. Jedoch lässt ein Vergleich der Datenraten, mit denen Cache-Hits und Cache-Misses übertragen werden, unter Berücksichtigung der Objekt-Trefferrate eine Abschätzung zu.

In Kapitel 2.2.2.5 wird eine differenziertere Betrachtung von Cache-Hits und Cache-Misses vorgenommen. Die Anwendung der daraus abgeleiteten Berechnungen erfolgt bei der Analyse von Cache-Verbünden in Kapitel 3.

2.2.2.2 Nutzung

Ein WWW-Cache-Server erbringt erst dann einen Nutzen, wenn mehrere Nutzer dieselben Inhalte aus dem World Wide Web abrufen. Die Wahrscheinlichkeit dafür, dass Objekte mehrfach abgerufen werden, steigt mit zunehmender Anzahl Nutzer [DMF97]. Daher ist eine zentrale Fragestellung bei dem Einsatz eines Cache-Servers, durch welche Maßnahmen möglichst viele Nutzer über den Cache-Server geleitet werden können. Es lassen sich drei Ansätze unterscheiden [Gri98a]:

Freiwillige Nutzung
: Den Nutzern wird freigestellt, den WWW-Cache-Server zu verwenden. Die freiwillige Nutzung wird sich nur dann einstellen, wenn der WWW-Cache-Server durch Verringerung der Wartezeiten einen erkennbaren Vorteil erbringt.

Erzwungene Nutzung durch Sperrung von ausgehendem HTTP-Verkehr
: Auf einem Router kann ausgehender WWW-Verkehr anhand des Ziel-Ports 80 für HTTP identifiziert werden. Unter Verwendung entsprechender Kontrollmechanismen auf dem Router ist es möglich, lediglich dem Cache-Server direkten Zugriff auf WWW-Server zu gestatten. Ausgehender WWW-Verkehr von ande-

ren Systemen im Netz wird durch den Router blockiert. Dadurch werden die Nutzer gezwungen, ihren WWW-Browser zur Nutzung des Cache-Servers zu konfigurieren.

Erzwungene Nutzung durch transparente Umleitung
Ein Router im lokalen Netz identifiziert ausgehenden WWW-Verkehr und leitet ihn automatisch an einen Cache-Server um. Dieses Verfahren ist aus Sicht der Nutzer transparent, da sie keine Konfiguration an ihrem WWW-Browser vornehmen müssen. Für die Umleitung und die Kommunikation zwischen Router und Cache-Server stehen Protokolle, wie das *Web Cache Coordination Protocol* (WCCP) von Cisco [CFTW01], zur Verfügung.

Eine wesentliche Schwierigkeit bei der freiwilligen Nutzung liegt darin, dass unzufriedene Nutzer den direkten Weg zu den WWW-Servern vorziehen und kaum wieder auf den Cache-Server zurückgreifen werden. Ein langfristiger Nutzen des Cache-Servers ist hierbei fraglich. Demgegenüber liegt bei erzwungenem Caching ein Problem in der Verfügbarkeit des Cache-Servers. Ohne funktionierenden Cache-Server bleibt der Zugang zum World Wide Web versperrt. Zudem muss der Cache-Server den gesamten aufkommenden WWW-Verkehr ohne Beeinträchtigung der Performance bearbeiten können.

2.2.2.3 Aufbau

In Abbildung 2.8 sind die Komponenten dagestellt, die die zentralen Aufgaben in einem Cache-Server übernehmen [Dan98].

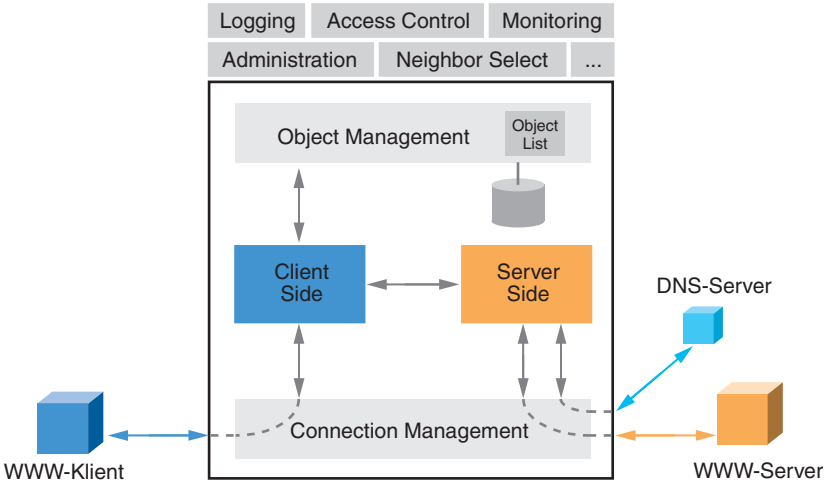


Abbildung 2.8: Komponenten eines Cache-Servers

Client-Side und Server-Side arbeiten auf der Ebene der Anwendungsprotokolle. Die Client-Side nimmt die Requests der Klienten entgegen und leitet entsprechende Responses zurück. Die Implementierung zur Bearbeitung von HTTP-Verkehr gleicht

im Prinzip auf der Client-Side der eines WWW-Servers. Umgekehrt werden auf der Server-Side die Protokolleigenschaften eines WWW-Klienten nachgebildet. Sollen neben HTTP auch weitere Protokolle, wie z. B. FTP oder SSL, von dem Cache-Server unterstützt werden, müssen die entsprechenden Implementierungen auf Client- und Server-Side vorliegen. Darüber hinaus müssen auf der Server-Side DNS-Abfragen ausgeführt werden, um die symbolischen Namen der Server auf IP-Adressen abzubilden.

Sämtliche Verbindungen zwischen dem Cache-Server und Klienten oder Servern werden von dem Connection Management verwaltet. Es stellt die Zuordnung zu den entsprechenden Instanzen auf Client-Side oder Server-Side sicher und regelt das Vorgehen bei Timeouts oder unerwartet abgebrochenen Verbindungen.

Das Object Management steuert das Auslesen und Speichern der Objekte auf der Festplatte des Cache-Servers. Für jeden eingehenden HTTP-Request wird das Object Management von der Client-Side aufgerufen, um die Existenz und Aktualität des geforderten Objekts im Speicher zu prüfen. Ist das Objekt veraltet oder nicht vorhanden, muss über das Connection Management und die Server-Side eine Verbindung zum originalen WWW-Server aufgebaut werden. Das Object Management führt eine Object List, in der die gespeicherten Objekte verzeichnet sind. Für eine effiziente Suche werden die URLs in der Object List nicht im Klartext, sondern durch Hashwerte, wie z. B. einen 16 Byte breiten MD5-Schlüssel [Ri92], dargestellt. Diese Darstellung bringt gleichzeitig den Vorteil, dass die Schlüssel auch als Dateinamen konstanter Länge für die Speicherung der Objekte verwendet werden können. Über den Einsatz geeigneter Strategien zur Ersetzung von Objekten in dem Speicher liegen in der Literatur zahlreiche Untersuchungen vor (u. a. [DiAr99] [AFJ98]). Bei ausreichend großem Speicher bieten sich einfach zu implementierende Verfahren, wie z. B. *Least Recently Used* (LRU), an [Wes01a].

Neben den genannten notwendigen Komponenten verfügt ein Cache-Server in der Regel über eine Reihe weiterer Funktionen. So wird das Ergebnis einer bearbeiteten Anfrage unter Angabe entsprechender Statusinformationen in einer Logdatei festgehalten. Durch Analyse der Logdatei können Trefferraten berechnet oder Ursachen auftretender Probleme ermittelt werden. Eine weitere typische Funktion stellt die Zugriffskontrolle dar, mit der der Kreis der Nutzer eines Cache-Servers eingeschränkt werden kann. Die Zugriffskontrolle ist eine Erweiterung des Connection Managements. Als weitere Ergänzungen lassen sich u. a. Funktionen für das Konfigurations-Management oder die Systemüberwachung nennen.

Bei der technischen Umsetzung lassen sich hardware-basierte und software-basierte Cache-Server unterschieden. In hardware-basierten Cache-Servern sind sämtliche Komponenten einschließlich Hardware und Betriebssystem für den Einsatz als Cache-Server optimiert [TML99]. Diese Geräte können nur als Cache-Server betrieben werden.

Ein software-basierter Cache-Server besteht aus einem herkömmlichen UNIX-basierten System, auf dem eine eigenständige Caching-Software installiert ist. Software-basierte Cache-Server weisen in der Regel geringere Leistungswerte als hardware-basierte auf. Demgegenüber sind die Anschaffungskosten von software-basierten Caches-Servern geringer, und sie können außerdem für andere Zwecke, so z. B. als Datenbank- oder WWW-Server, verwendet werden.

2.2.2.4 Performance

In der Literatur finden sich zahlreiche Untersuchungen über die Performance von WWW-Servern (u. a. [NBK02] [AAY97] [BDR97] [BaCr98] [HaDi97]) und Cache-Servern (u. a. [MEW00] [RoSo99] [FCD+99]). Es ist zu beachten, dass sich Untersuchungen über WWW-Server nur bedingt auf Cache-Server übertragen lassen. Ein WWW-Server stellt eine eindeutige Datenquelle im Netz dar. In dieser Datenquelle werden Objekte von den Festplatten gelesen, eventuell im Hauptspeicher für weitere schnelle Zugriffe abgelegt und an die WWW-Klienten übertragen. Ein Cache-Server vereint sowohl Datenquelle als auch Datensenke. Das Lastverhalten eines Cache-Servers wird wesentlich durch häufige Lese- und Schreiboperationen auf den Festplatten bestimmt.

Zwei Maßzahlen können zur Beurteilung der Performance eines Cache-Servers herangezogen werden:

- Durchsatz, gemessen in HTTP-Responses / Sekunde,
- Antwortzeiten für Cache-Hits und Cache-Misses.

In den Antwortzeiten für Cache-Misses sind die Verzögerungszeiten enthalten, die durch die Verbindungen zu den WWW-Servern entstehen. Sie werden dennoch angegeben, um die Antwortzeiten der Cache-Hits besser beurteilen zu können. Häufig werden als weitere Maße Objekt- und Volumen-Trefferrate genannt. Da die Trefferraten jedoch von dem Speichervolumen eines Cache-Servers sowie von der Anzahl und der Zusammensetzung der Nutzer abhängen, dienen sie lediglich zur Beurteilung des Nutzens und nicht der Performance.

Für Aussagen über potentielle *Bottlenecks* von Cache-Servern können Erfahrungsberichte aus dem praktischen Einsatz herangezogen werden [Wes01a]. Bei der Hardware-Ausstattung beeinflussen I/O-Komponenten die Performance maßgeblich. Auch wenn bereits einfache Festplattensysteme bis zu einer theoretischen Grenze von 100 MByte/s arbeiten, verringern die häufigen Lese- und Schreiboperationen für kleine Dateien diese Rate für einen Cache-Server auf maximal 10 MByte/s. Andere Komponenten, wie Prozessor, Hauptspeicher oder Netzinterface, haben dagegen keinen signifikanten Einfluss oder können ohne großen Aufwand den Anforderungen entsprechend ausgelegt werden.

Die Unterschiede zwischen hardware- und software-basierten Cache-Servern werden bei einer Betrachtung der Betriebssysteme deutlich. Software-basierte Cache-

Server verwenden das auf die jeweilige Hardware angepasste UNIX-Betriebssystem. Da UNIX-Betriebssysteme für einen Mehrbenutzerbetrieb ausgelegt sind, liegen die zentralen Aufgaben in einer sicheren Zugriffssteuerung auf die Hardware-Ressourcen oder das Dateisystem sowie in einem zuverlässigen Prozess-Management [Bac86]. Diese Zielsetzung geht jedoch zu Lasten der Performance [Mog95b] [BDM98]. Als deutlicher Engpass hat sich dabei das *Standard UNIX File System* (UFS), bzw. betriebssystem-abhängige Varianten davon, erwiesen. So kann allein durch den Einsatz eines alternativen, für Caching optimierten Dateisystems der Durchsatz eines software-basierten Cache-Servers um den Faktor 2–4 erhöht werden [SGHS01].

Neben dem Dateisystem hat auch der TCP/IP-Stack des Betriebssystems entscheidenden Einfluss auf die Performance [BDM99]. Wie bereits in Kapitel 2.1.3.4 erläutert, lässt sich der TCP/IP-Stack durch Verändern der entsprechenden Kernelparameter für den Einsatz in lokalen oder Weitverkehrsnetzen optimieren.

Fast alle Hersteller von hardware-basierten Cache-Servern verwenden eigene Microkernel-Architekturen [BaOb00]. Ein Betriebssystem lässt sich wesentlich vereinfachen, indem auf kooperative Mechanismen, die die Zugriffe verschiedener Prozesse auf die Ressourcen des Systems regeln, verzichtet wird. Dem Cache-Prozess als zentraler Instanz können alleinige oder zumindest priorisierte Zugriffsrechte auf die Hardware-Ressourcen und das Dateisystem eingeräumt werden [TML99].

Zur Messung der Performance von Cache-Servern hat sich die Software *Polygraph* etabliert, die sowohl WWW-Klienten als auch WWW-Server nachbildet [Pol01]. Durch Angabe von u. a. Objekt-Trefferraten, Objektgrößen oder Zwischenankunftszeiten erzeugen die WWW-Klienten einen charakteristischen Strom von HTTP-Requests. Die WWW-Server senden entsprechende HTTP-Responses zurück, wobei sie durch einstellbare Verzögerungen oder Paketverluste die Datenraten beeinflussen. Die daraus resultierende Workload bildet den typischen WWW-Verkehr in Weitverkehrsnetzen nach.

In regelmäßig stattfindenden *Cache-Offs* stellen Hersteller von Cache-Servern ihre Geräte für Benchmarks zur Verfügung, um unter identischen Voraussetzungen vergleichbare Resultate zu erzielen. Der letzte Cache-Off fand im Oktober 2000 statt [RoWe00]. Leistungsfähige Cache-Server erreichten dabei einen Durchsatz von über 2.000 HTTP-Responses/s. Bei den meisten Cache-Servern lagen die mittleren Anwortzeiten für einen Hit unter 200 ms, für einen Miss unter 3 s.

Allgemein lassen diese Benchmarks erkennen, dass hardware-basierte Cache-Server deutlich leistungsfähiger sind als software-basierte. Hardware-basierte Cache-Server werden jedoch nur in geringen Stückzahlen gefertigt und sind entsprechend teuer. Deshalb zeigt eine Normierung der Ergebnisse auf einen einheitlichen Preis, dass auch der Einsatz software-basierter Cache-Server unter entsprechenden Voraussetzungen gerechtfertigt ist.

2.2.2.5 Konsistenz zwischen Cache-Server und WWW-Server

Seit HTTP/1.0 stehen Verfahren zur Verfügung, mit denen Cache-Server die Aktualität der gespeicherten Objekte überprüfen können. So enthält eine HTTP-Response in dem Entity-Header `Last-Modified: "date"` das Datum der letzten Änderung eines Objekts. Unter Verwendung dieses Datums, das einer Versionsnummer des Objekts entspricht, kann der Cache-Server in einem *bedingten* HTTP-Request die Überprüfung durch den WWW-Server veranlassen. Hierfür wird im Header des bedingten HTTP-Requests die Zeile `If-Modified-Since: "date"` (IMS) eingefügt. Ist das Objekt im Cache-Server noch aktuell, sendet der WWW-Server eine HTTP-Response mit der Status-Line `HTTP/1.0 304 Not Modified` zurück. Der Cache-Server kann nach Erhalt dieser Antwort das Objekt aus dem Speicher an den WWW-Klienten übertragen. Ist das Objekt auf dem Cache-Server dagegen veraltet, antwortet der WWW-Server mit der Übertragung des aktuellen Objekts. Die Übertragung des Objekts wird, wie nach dem Empfang eines herkömmlichen HTTP-Requests, mit der Status-Line `HTTP/1.0 200 OK` eingeleitet.

Die Einhaltung einer *starken* Konsistenz erfordert, dass bei jedem Cache-Hit zunächst die Aktualität des Objekts geprüft wird. Eine Überprüfung kann jedoch nur durch einen HTTP-Request nach dem oben dargestellten Verfahren an den originalen WWW-Server erfolgen, wodurch signifikante Verzögerungen entstehen. Deshalb steht eine starke Konsistenz im Widerspruch zu der Forderung, dass Cache-Server die Antwortzeiten im World Wide Web reduzieren sollen [LiCa97]. Um die Überprüfung für jeden Cache-Hit zu vermeiden, werden auf Cache-Servern heuristische Verfahren zur Abschätzung der Aktualität verwendet. Mit diesem Verfahren lässt sich jedoch nur eine *schwache* Konsistenz erreichen, bei der keine Garantie für die Aktualität der Inhalte gegeben werden kann [GwSe96].

Durch Einführung weiterer Header in HTTP/1.0 sollte die schwache Konsistenz auf Cache-Servern zuverlässiger gestaltet werden. So gibt der Entity-Header `Expires: "date"` in einer HTTP-Response ein festes Datum an, an dem ein Cache-Server spätestens die Aktualität des Objekts auf dem WWW-Server überprüfen muss. Der Nutzen dieses Headers ist jedoch begrenzt, da nur selten a priori bekannt ist, wann ein Objekt geändert werden wird. Außerdem werden Cache-Server das Objekt nicht abrufen, auch wenn es wider Erwarten doch vor dem mit `Expires:` angegebenen Datum auf dem WWW-Server geändert wird. Es zeigte sich, dass dieser Header hauptsächlich verwendet wird, um durch Angabe eines bereits vergangenen Datums den Cache-Server zu einer ständigen Aktualisierung des Objekts zu zwingen [WiMi99]. Für einen ähnlichen Zweck wurde in HTTP/1.0 ein weiterer Entity-Header `Pragma: no-cache` eingeführt, mit dem das Speichern von Objekten auf einem Cache-Server sogar völlig unterbunden werden kann.

Mit HTTP/1.1 wurde der Entity-Header `Age: "value"` eingeführt, durch den ein Cache-Server Klienten signalisiert, vor wie viel Sekunden die Aktualität des übertragenen Objekts auf dem originalen WWW-Server überprüft wurde. Als weitere

Ergänzung steht in HTTP/1.1 der General-Header `Cache-Control: "value"` für HTTP-Requests und HTTP-Responses zur Verfügung. Durch die Angabe verschiedener Direktiven in diesem Header lässt sich die Speicherung von Objekten in Caches genauer steuern. Unter anderem stehen folgende Direktiven zur Verfügung:

- `private`: das Speichern des Objekts in der HTTP-Response ist nur im Browser-Cache, nicht aber auf Cache-Servern gestattet.
- `no-store`: das Speichern des Objekts in der HTTP-Response ist sowohl im Browser-Cache als auch auf Cache-Servern untersagt.
- `no-cache`: das Speichern des Objekts in der HTTP-Response ist sowohl im Browser-Cache als auch auf Cache-Servern gestattet. Vor der Auslieferung aus dem Cache muss jedoch stets die Aktualität des Objekts überprüft werden.
- `max-age=n`: gibt in einer HTTP-Response an, wie viele Sekunden `n` das Objekt von dem Cache-Server ohne vorhergehende Aktualisierung ausgeliefert werden darf. Die Wirkung entspricht dem Entity-Header `Expires:`. In einem HTTP-Request signalisiert `max-age=n` dem Cache-Server, dass der WWW-Klient nur solche Objekte akzeptiert, die vor weniger als `n` Sekunden aktualisiert worden sind.

In Ergänzung zu Kapitel 2.2.2.1 lässt sich die Definition eines Cache-Hits ausweiten. So liegt ein *unbestätigter* Cache-Hit vor, wenn der Cache-Server anhand der Heuristik das Objekt als aktuell beurteilt. Wird das Objekt dagegen als veraltet beurteilt, wird es mit einem IMS-Request an den WWW-Server überprüft. Sendet der WWW-Server ein `Not-Modified` zurück, handelt es sich um einen *bestätigten* Cache-Hit und das Objekt kann aus dem Cache-Server übertragen werden. Wird ein aktuelles Objekt von dem WWW-Server übertragen, handelt es sich um einen bestätigten Cache-Miss. In diesem Fall wird das Objekt vom WWW-Server an den WWW-Klienten weitergeleitet, wobei das veraltete Objekt im Cache-Server mit dem aktuellen überschrieben wird. Eine Übersicht der erweiterten Definitionen von Cache-Hit und Cache-Miss ist in Tabelle 2.1 zusammengefasst.

Hit / Miss	Objekt ist im Cache	Objekt ist aktuell	HTTP-Request an WWW-Server	HTTP-Response von WWW-Server
unbestätigter Hit	ja	ja	–	–
bestätigter Hit	ja	nein	IMS	`304 Not Modified`
bestätigter Miss	ja	nein	IMS	`200 OK`
einfacher Miss	nein	–	GET	`200 OK`

Tabelle 2.1: Erweiterte Definition von Cache-Hit und Cache-Miss

Den höchsten Nutzen erbringen unbestätigte Cache-Hits, da die Objekte ohne vorhergehende Überprüfung auf dem WWW-Server sofort ausgeliefert werden können. Das Datenaufkommen, das bei bestätigten Cache-Hits zwischen Cache-Server und

WWW-Server anfällt, ist vernachlässigbar gering. Die Zeit, die bei der Überprüfung des Objekts anfällt, verzögert jedoch die Auslieferung eines bestätigten Cache-Hits gegenüber einem unbestätigten Cache-Hit deutlich. Bei bestätigten Cache-Misses und einfachen Cache-Misses werden weder Datenvolumen noch Übertragungszeit eingespart.

Unbestätigte Cache-Hits verursachen keinen Datenverkehr zwischen Cache-Servern und WWW-Servern. Hierdurch ergibt sich der hohe Nutzen von unbestätigten Cache-Hits, jedoch auf Kosten einer schwachen Konsistenz. Bei bestätigten Cache-Hits, bestätigten Cache-Misses und einfachen Cache-Misses wird dagegen die Konsistenz im Rahmen der möglichen Genauigkeit gewahrt. Daraus folgt, dass eine Erhöhung der Trefferraten durch unbestätigte Cache-Hits zu Lasten einer gesicherten Konsistenz geht. Nach welchen Verfahren die Aktualität von Objekten abgeschätzt werden kann, um auch bei unbestätigten Cache-Hits eine möglichst sichere Konsistenz zu gewährleisten, wird im folgenden Kapitel untersucht.

2.2.2.6 Heuristik zur Beurteilung der Aktualität

Die Heuristik zur Beurteilung der Aktualität eines Objekts auf einem Cache-Server ist nicht Bestandteil einer HTTP-Spezifikation. Im Folgenden wird beispielhaft der Algorithmus erläutert, der in der Cache-Software *Squid* implementiert ist [Wes01b].

Zur Berechnung der Aktualität werden für jedes Objekt zunächst folgende Variablen definiert:

- `NOW`: aktuelles Datum auf dem Cache-Server bei Eintreffen eines HTTP-Requests.
- `EXPIRES`: das im Entity-Header `Expires:` der HTTP-Response angegebene Datum.
- `OBJ_DATE`: das im General-Header `Date:` der HTTP-Response angegebene Datum des WWW-Servers.
- `OBJ_LASTMOD`: das im Entity-Header `Last-Modified:` der HTTP-Response angegebene Datum der letzten Änderung des Objekts.
- `CLIENT_MAX_AGE`: die im General-Header `Cache-Control: max-age=n` angegebene maximal akzeptierbare Verweildauer des Objekts im Cache-Server.

Sämtliche Zeitinformationen werden mit einer Auflösung von 1 Sekunde angegeben. Die Verwendung der Header `Date:` und `Last-Modified:` wird von den HTTP-Spezifikationen nicht zwingend gefordert. Fehlen diese Informationen in einer HTTP-Response, werden unter Annahme des *worst case* folgende Ersetzungen in Squid vorgenommen:

- Ein fehlender General-Header `Date:` wird durch das aktuelle Datum auf dem Cache-Server ersetzt, an dem das Ende des Headers der HTTP-Response gelesen wird.

- Ein fehlender Entity-Header `Last-Modified:` wird durch den General-Header `Date:` aus derselben HTTP-Response ersetzt (`OBJ_LASTMOD=OBJ_DATE`).

Außerdem wird für den Algorithmus vorausgesetzt, dass die zeitlich kausale Reihenfolge `OBJ_LASTMOD` ≤`OBJ_DATE` ≤`NOW` eingehalten wird. Bei einer Verletzung dieser Forderung werden die Ersetzungen `OBJ_DATE=NOW` sowie `OBJ_LASTMOD=OBJ_DATE` vorgenommen.

Aus den genannten Variablen werden in einem nächsten Schritt folgende Größen abgeleitet:

- `LM_AGE=OBJ_DATE-OBJ_LASTMOD`: das Alter des Objekts auf dem WWW-Server zum Zeitpunkt der letzten Aktualisierung.
- `OBJ_AGE=NOW-OBJ_DATE`: die bisherige Verweildauer des Objekts auf dem Cache-Server seit der letzten Aktualisierung.
- `LM_FACTOR=OBJ_AGE/LM_AGE`: Verhältnis von Alter des Objekts auf dem WWW-Server zur Verweildauer des Objekts im Cache-Server.

In Abbildung 2.9 sind die zuvor definierten Variablen dargestellt. Weiterhin sind drei mögliche Zeitpunkte eingefügt, wie sie im Entity-Header `Expires:` angegeben werden können. So liegt `EXPIRES_1` vor `OBJ_DATE` und damit auch stets vor `NOW`. Dieses Objekt muss daher bei jedem Zugriff auf Aktualität geprüft werden. Der Zeitraum zwischen `OBJ_DATE` und `EXPIRES_2` ist zum Zeitpunkt `NOW` überschritten, so dass auch dieses Objekt aktualisiert werden muss. Das Objekt, dessen Verweildauer erst zum Zeitpunkt `EXPIRES_3` abläuft, kann hingegen unmittelbar ausgeliefert werden.

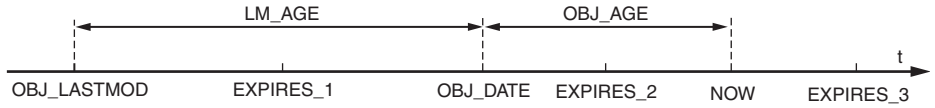


Abbildung 2.9: Variablen zur Berechnung der Aktualität

Zur Steuerung des Algorithmus lassen sich drei Parameter, die so genannten *Refresh Pattern*, konfigurieren:

- `CONF_MIN`: untere Grenze für die Verweildauer der Objekte im Cache-Server. Der Standardwert ist 0.
- `CONF_MAX`: obere Grenze für die Verweildauer der Objekte im Cache-Server. Der Standardwert beträgt 259.200 Sekunden bzw. 3 Tage.
- `CONF_PERCENT`: obere Grenze für das Verhältnis `LM_FACTOR`. Der Standardwert beträgt 0,2, d. h. die zulässige Verweildauer der Objekte im Cache-Server beträgt 1/5 des Alters zum Zeitpunkt der Auslieferung von den WWW-Servern.

Der vollständige Algorithmus ergibt sich aus Abbildung 2.10. Zu beachten ist die Reihenfolge der Bearbeitung. Falls vorhanden, haben die Angaben der Header `Expires:` oder `Cache-Control:` Priorität. Erst danach werden die Refresh Pattern

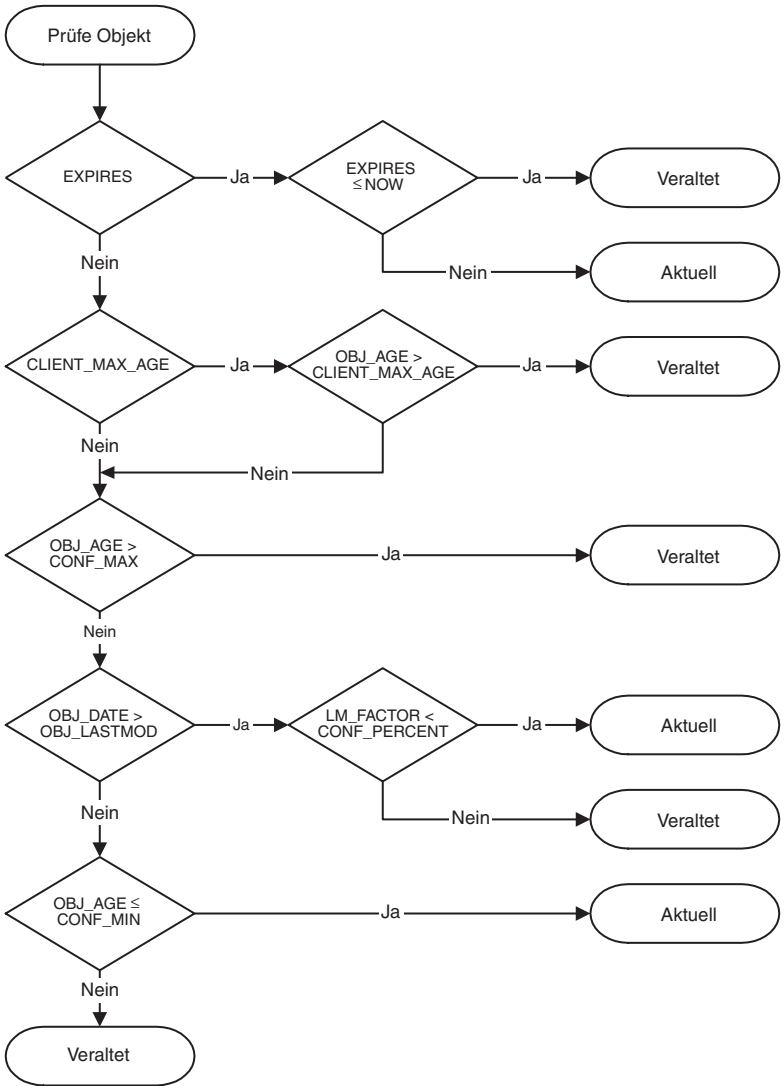


Abbildung 2.10: Algorithmus zur Überprüfung der Aktualität

verwendet, wobei zuerst die Überschreitung der maximalen Verweildauer, dann das Verhältnis `LM_FACTOR` und schließlich die minimale Verweildauer im Cache-Server geprüft wird.

Verwendung und Zuverlässigkeit von Zeitinformationen in HTTP

Im Vorfeld dieser Arbeit wurden Untersuchungen über die Verwendung und die Zuverlässigkeit von Zeitinformationen in HTTP durchgeführt. Eine erste Untersuchung ergab, dass im World Wide Web keine hohe Synchronität der Rechneruhren vorausgesetzt werden darf (s. Anhang A.1). In HTTP/1.1 wird die Synchronisierung der Uhren über das *Network Time Protocol* (NTP) [Mil92] gefordert, wodurch die Differenz zwischen zwei Rechneruhren auf unter 1 Sekunde reduziert werden könn-

te. Dennoch wurden in der Untersuchung 20% WWW-Server mit einem Offset von über 10 Minuten festgestellt. Daraus folgt, dass bei einer Auswertung der Zeitangaben in HTTP-Headern potentiell signifikante Abweichungen von Rechneruhren berücksichtigt werden müssen.

Die Ungenauigkeit der Zeitangaben führt dazu, dass die Verwendung fester Zeitpunkte bei der Beurteilung der Aktualität von Objekten vermieden werden sollte [Mog99]. So wird die Nutzung des Headers `Expires:`, der einen festen Zeitpunkt enthält, nicht mehr empfohlen, da für die korrekte Behandlung die Synchronität der Uhren auf WWW-Server und Cache-Server erforderlich ist. Stattdessen wird der Header `Cache-Control: max-age=n` mit der Angaben eines Zeitraumes empfohlen.

Die Heuristik in Squid verwendet mittlerweile ebenfalls Zeitdifferenzen, bei denen ein Offset zwischen WWW-Server und Cache-Server unerheblich ist. So ist das Alter `LM_AGE` bei der Auslieferung eines Objekts die Differenz von zwei Zeitpunkten auf dem WWW-Server. Die Verweildauer `OBJ_AGE` des Objekts im Cache-Server wird zwar in dem dargestellten Verfahren aus je einem Zeitpunkt auf WWW-Server und Cache-Server gebildet. Offenbar wird jedoch in neuen Implementierungen statt der Zeit aus dem Header `Date:` die aktuelle Zeit auf dem Cache-Server bei dem Eintreffen der HTTP-Response gewählt. Für eine genau Darstellung der Implementierung müsste in dem Algorithmus zwischen `OBJ_DATEWWW-Server` und `OBJ_DATECache-Server` unterschieden werden.

Bedeutung der Refresh Pattern

In einer zweiten Untersuchung wurden die Angaben in HTTP-Responses betrachtet, die die Speicherung der Objekte auf Cache-Servern steuern und, unter Berücksichtigung der Refresh Pattern, die Häufigkeit von Aktualisierungen beeinflussen (s. Anhang A.2). Die Untersuchung ergab, dass bei 3,2% der Objekte die Speicherung auf einem Cache-Server untersagt wird. Weitere 11,1% lassen sich zwar speichern, müssen jedoch stets vor einer erneuten Auslieferung aktualisiert werden. Demgegenüber können 70% der Objekte für den vollständigen Zeitraum von 3 Tagen, der durch den Parameter `CON_MAX` festgelegt wird, ohne vorhergehende Aktualisierung ausgeliefert werden. Für die restlichen 15% wird anhand des Parameters `CONF_PERCENT` festgelegt, ob eine Aktualisierung notwendig ist.

In einem weiteren Schritt wurde die Auswirkung des Parameters `CONF_PERCENT` auf die Häufigkeit der Aktualisierungen untersucht. Unter Berücksichtigung der Standardwerte der Refresh Pattern beeinflusst dieser Parameter die Verweildauer von nur 15% der Objekte. Dieser Anteil erhöht sich erst dann signifikant, wenn der Parameter auf geringe Werte unterhalb `0.1` eingestellt wird.

Mit den genannten Untersuchungen wurde der Einfluss betrachtet, den Refresh Pattern und Angaben in HTTP-Headern auf die Häufigkeit der Aktualisierungen von Objekten ausüben. Diese Betrachtungen sind notwendig, da die Häufigkeit der Ak-

tualisierungen direkten Einfluss auf die Einsparungen hat, die mit Cache-Servern erreicht werden können. So führen hohe Verweildauern zu einem Anstieg der unbestätigten Cache-Hits. Aus den Untersuchungen wird deutlich, dass eine Erhöhung der Refresh Pattern `CONF_PERCENT` sowie `CONF_MAX` zu einer höheren Trefferrate führt.

Umgekehrt führt eine geringe Verweildauer zu häufigen Aktualisierungen und damit zu einem Anstieg der bedingten Cache-Hits oder bestätigten Cache-Misses. Allerdings ist anzunehmen, dass eine geringe Verweildauer eine bessere Konsistenz der Objekte sichert. Wieweit jedoch Veränderungen der Refresh Pattern tatsächlich die Konsistenz der Objekte beeinflussen, kann mit diesen Untersuchungen nicht beantwortet werden. Hierfür müsste eine ständige Überprüfung der Objekte auf den WWW-Servern durchgeführt werden, was jedoch aufgrund des hohen Aufwandes kaum durchführbar ist. Entsprechende Untersuchungen sind auch in der Literatur nicht bekannt.

2.2.2.7 Umgang mit Cookies

HTTP wurde als statusfreies Protokoll konzipiert, in dem der WWW-Server HTTP-Requests unabhängig voneinander bearbeitet. Eine Reihe von Anwendungen ist jedoch darauf angewiesen, dass Statusinformationen über mehrere Verbindungen weitergegeben und gegebenenfalls über längere Zeiträume gespeichert werden können. Mit Hilfe von CGI-Programmen lassen sich lediglich einfache Anwendungen, wie z. B. das Ausfüllen einer Reihe von Formularen, realisieren. Erst seit der Einführung der so genannten *Cookies* können vollständige Statusinformationen zwischen aufeinander folgenden HTTP-Requests übertragen und sogar über verschiedenen Sitzungen gespeichert werden [KrMo00].

Cookies werden ausschließlich auf WWW-Servern generiert. Für die Übertragung an einen WWW-Klienten enthält die HTTP-Response den Header `Set-Cookie: "value"`. Der Wert des Cookies besteht aus einem beliebigen ASCII-Text von maximal 4 KByte Länge. Der WWW-Klient legt den Inhalt des Cookies zusammen mit dem URL des Objekts für spätere Zugriffe auf der Festplatte ab. Bei dem nächsten Zugriff auf das Objekt erweitert der WWW-Klient unter Verwendung des gespeicherten Cookies den Header des HTTP-Requests um das Feld `Cookie: "value"`. Der WWW-Server kann das geforderte Objekt in Abhängigkeit des Cookies generieren.

Ein Cache-Server darf Objekte mit Cookies nicht ohne zusätzliche Maßnahmen speichern und an andere Klienten ausliefern. In der Cache-Software Squid wird vor der Speicherung der Eintrag `Set-Cookie: "value"` aus dem HTTP-Header eines Objekts gelöscht. HTTP-Requests nach dem Objekt werden mit einem Header `If-modified-since:` an den WWW-Server gesendet. Cookies in HTTP-Requests werden nicht verändert.

Die Tatsache, dass Cookies unbestätigte Cache-Hits verhindern, scheint zunächst den Nutzen von Cache-Servern zu mindern. Verschiedene Untersuchungen zeigen jedoch, dass der Einsatz von Cookies derzeit auf weniger als 5% der Objekte beschränkt ist [Wes01a] [WiMi99].

2.3 Cache-Verbünde

Die vorangegangene Betrachtung von Cache-Servern beschränkt sich auf lokale Netze, in denen die Abfragen mehrerer WWW-Klienten über einen lokalen Cache-Server geleitet werden. Diese Betrachtung lässt sich analog auf eine Umgebung ausweiten, in der mehrere lokale Cache-Server einen zentralen Cache-Server nutzen. Die gesamte Nutzen einer solchen Cache-Hierarchie setzt sich aus den Einsparungen der lokalen und zentralen Cache-Server zusammen. Der Einsatz zentraler Cache-Server bietet sich für größere Organisationen an, in denen bereits mehrere lokale Cache-Server betrieben werden. Eine weitere Möglichkeit stellt der Betrieb in Kernnetzen dar, in denen zentrale Cache-Server an Übergabepunkten zu anderen Netzen oder vor besonders ausgelasteten Netzsegmenten, wie z. B. Transatlantikstrecken, installiert werden.

Die Bildung von Cache-Hierarchien stellt eine Reihe neuer Anforderungen an die Cache-Server. Die bisher betrachteten Cache-Server verfügen zunächst über keine eigenständigen Mechanismen zur Unterstützung von Cache-Hierarchien. Eine mögliche Erweiterung der Cache-Server besteht darin, dass sämtliche Anfragen, die keinen unbestätigten Cache-Hit ergeben, an einen übergeordneten Cache-Server weitergeleitet werden. Mit diesem Ansatz lassen sich Cache-Hierarchien bilden, deren Topologie durch eine baumförmige Struktur gekennzeichnet ist (Abbildung 2.11 links).

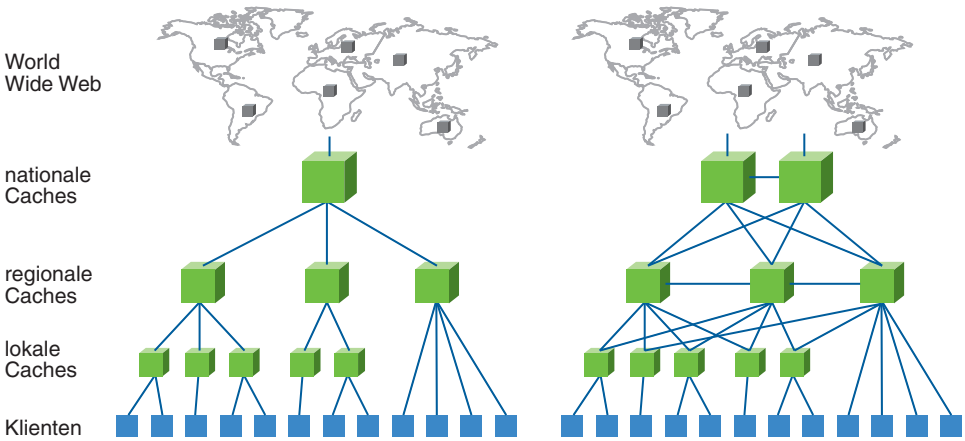


Abbildung 2.11: Hierarchien von Cache-Servern

Baumförmige Strukturen eignen sich aus verschiedenen Gründen nicht für die Bildung leistungsfähiger Cache-Hierarchien in Netzen mit hohem Verkehrsaufkommen:

- Jeder übergeordnete Cache-Server stellt aus Sicht seiner Klienten einen so genannten *single point of failure* dar.
- Die Lastanforderungen an Cache-Server steigen mit höheren Ebenen der Cache-Hierarchie. Es besteht keine Möglichkeit, die Last auf mehrere Cache-Server zu verteilen.
- Die Verkehrsflüsse innerhalb der Cache-Hierarchie sind durch die Topologie fest vorgegeben. Eine dynamische Verkehrslenkung in Abhängigkeit von der aktuellen Lastsituation ist nicht möglich.

Ausgehend von diesen Überlegungen erfordern leistungsfähige Cache-Hierarchien Verfahren, mit denen sich Topologien wie in Abbildung 2.11 rechts bilden lassen. In dieser Topologie wird die strenge Hierarchie zugunsten einer verteilten Struktur aufgelöst. Eine notwendige Voraussetzung für die Bildung solcher Cache-Verbünde ist die Möglichkeit, dass einem Cache-Server mehrere übergeordnete Cache-Server zur Verfügung stehen. Weiterhin müssen über geeignete Mechanismen Zustandsinformationen und Informationen über die Speicherinhalte zwischen Cache-Servern ausgetauscht werden [DHS93].

Ausgelöst durch das Harvest-Projekt [BDH+94] der University of South Colorado, Boulder, wurde Anfang 1995 der *Harvest Object Cache* entwickelt. Ein wesentliches Merkmal dieser Cache-Software war die Implementierung eines eigenständigen Protokolls für die Kommunikation zwischen Cache-Servern [CDN+96]. Mit dem später als *Internet Cache Protocol* (ICP) bezeichneten Protokoll wurde es möglich, Cache-Server zu beliebigen Strukturen miteinander zu koppeln. Zusammen mit ICP wurden grundlegende Konzepte für die Kommunikationsbeziehungen innerhalb von Cache-Verbünden eingeführt.

Nach Beendigung des Projektes zerfiel die weitere Entwicklung des Harvest Object Cache in zwei Richtungen. Unter dem Namen Harvest Cache entstand eine kommerzielle Cache-Software der Firma NetCache (heute Network Appliance) [Net01], wogegen aus einem neuen Projekt des National Laboratory for Applied Network Research (NLANR) die frei verfügbare Cache-Software *Squid* stammt [Wes01b]. In beiden Entwicklungen wurden die Neuerungen des Harvest Object Cache zur Bildung leistungsfähiger Cache-Verbünde fortgeführt.

Es ist zu beachten, dass ICP und neuere Protokolle zur Inter-Cache-Kommunikation lediglich der Übertragung von Zustandsinformationen und Informationen über die Speicherinhalte zwischen Cache-Servern dienen. Die Übertragung der WWW-Objekte erfolgt auch innerhalb eines Cache-Verbundes weiterhin über HTTP.

Der Darstellung der Inter-Cache-Protokolle wird zunächst eine Erläuterung der grundlegenden Kommunikationsbeziehungen innerhalb von Cache-Verbünden vorangestellt.

2.3.1 Kommunikationsbeziehungen in Cache-Verbünden

Mit dem Harvest Object Cache wurde ein zweistufiges Verfahren für die Kommunikation zwischen Cache-Servern vorgeschlagen. In diesem Verfahren wird davon ausgegangen, dass ein Cache-Server mit mehreren benachbarten Cache-Servern, den so genannten *Neighbors* (Nachbarn), kommuniziert. Die Neighbors werden in *Siblings* (Geschwister) und *Parents* (Eltern) unterschieden.

Durch Kommunikation über ein Inter-Cache-Protokoll erlangen die Cache-Server innerhalb eines Cache-Verbundes Kenntnis darüber, ob und auf welchen Neighbors ein gefordertes Objekt gespeichert ist. Das bei der Inter-Cache-Kommunikation übertragene Datenvolumen und die Häufigkeit der Anfragen hängen von den verwendeten Protokollen ab. Die Lenkung des HTTP-Verkehrs zwischen den Cache-Servern wird maßgeblich durch die Unterscheidung zwischen Siblings und Parent beeinflusst.

Sibling

Von einem Sibling fordert ein lokaler Cache-Server Objekte nur dann ab, wenn er a priori davon ausgehen kann, dass das Objekt auf dem Sibling vorhanden und aktuell ist. Dieser Regel entspricht, dass ein HTTP-Request von dem lokalen Cache-Server nach dem geforderten Objekt einen unbestätigten Cache-Hit auf dem Sibling erzeugt. Stellt der lokale Cache-Server fest, dass das geforderte Objekt auf keinem Sibling vorhanden hat, leitet er den HTTP-Request direkt an den originalen WWW-Server (Abbildung 2.12).

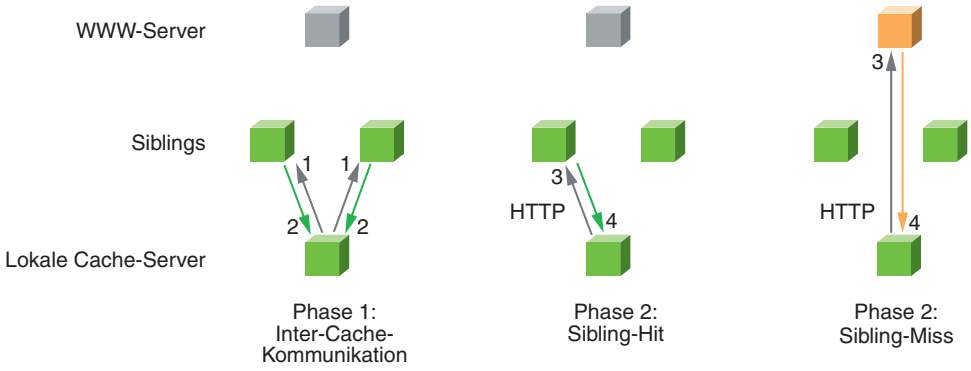


Abbildung 2.12: Kommunikationsbeziehung zwischen Cache-Server und Siblings

Parent

Im Gegensatz zu einem Sibling fordert der lokale Cache-Server ein Objekt von einem Parent auch dann ab, wenn das Objekt nicht auf dem Parent gespeichert ist. Um den HTTP-Request des lokalen Cache-Servers zu erfüllen, muss der Parent das Objekt von einem ihm übergeordneten Cache-Server oder dem originalen WWW-Ser-

ver abrufen. Dabei wird das Objekt auch auf dem Parent gespeichert (Abbildung 2.13).

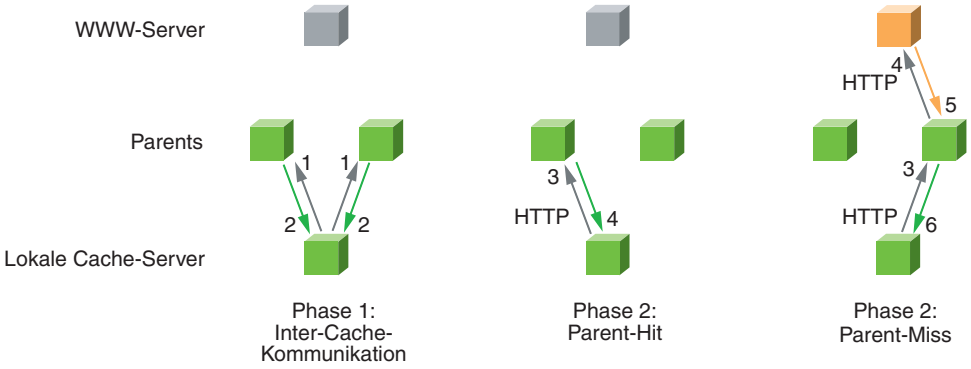


Abbildung 2.13: Kommunikationsbeziehung zwischen Cache-Server und Parents

Falsche Cache-Hits

Bei dem Abruf eines Cache-Hit von einem Neighbor kann ein zeitlich bedingter Zugriffskonflikt (*Race Condition*) auftreten. Hat ein lokaler Cache-Server die Information, dass das geforderte Objekt zum Zeitpunkt T_0 im Speicher des Neighbor vorhanden ist, sendet er einen HTTP-Request. Der Neighbor empfängt den HTTP-Request zum Zeitpunkt $T_0 + T_\Delta$. Es ist jedoch möglich, dass das Objekt innerhalb des Zeitraums T_Δ veraltet ist. Für die weitere Behandlung dieses so genannten falschen Cache-Hit (*false hit*) existieren drei unterschiedliche Möglichkeiten:

- Der Neighbor signalisiert dem lokalen Cache-Server mit der HTTP-Response, dass das Objekt mittlerweile veraltet ist. Anhand dieser Information wählt der lokale Cache-Server einen alternativen Neighbor aus oder befragt direkt den WWW-Server. Eine entsprechende Signalisierung ist in HTTP bisher nicht vorgesehen. Hierfür müssten entweder Erweiterungen von HTTP definiert oder Mechanismen der Inter-Cache-Kommunikation eingesetzt werden.
- Der Neighbor überprüft die Aktualität des Objekts gegenüber seinen Neighbors oder dem originalen WWW-Server. Anschließend wird das aktualisierte Objekt an den lokalen Cache-Server übertragen. Da hierbei eine Kommunikation zwischen dem übergeordneten Cache-Server und weiteren Instanzen erfolgt, entspricht dieses Verhalten dem eines Parent. Ist der Neighbor auf dem lokalen Cache-Servers lediglich als Sibling konfiguriert, wäre dieses Verhalten unter Umständen nicht erwünscht.
- Der Neighbor sendet das Objekt ohne vorhergehende Aktualisierung an den lokalen Cache-Server. Dieser Ansatz erfordert keine Erweiterungen, verschlechtert jedoch die Konsistenz bei großen T_Δ.

Nach welcher dieser drei Möglichkeiten ein falscher Cache-Hit aufgelöst wird, ist eine Frage der Implementierung oder der Konfiguration des Neighbor. In der Regel

wird empfohlen, dass sich der Neighbor bei einem falschen Cache-Hit wie ein Parent verhält und die Aktualität des Objekts überprüft, bevor er es an einen lokalen Cache-Server überträgt.

Bildung von Schleifen

In größeren Cache-Verbünden ist es bei unkoordinierter Konfiguration möglich, dass Schleifen (Loops) über mehrere Neighbors entstehen. Um diese Schleifen zu erkennen, hängt jeder Cache-Server seinen alphanumerischen Namen an den General-Header `Via:` einer HTTP-Response an:

```
Via: 1.1 frankfurt.cache.dfn.de:8080 (Squid/2.3.DEVEL4),
     1.1 hannover.cache.dfn.de:8080 (Squid/2.3.DEVEL4),
     1.1 www-cache.rrzn.uni-hannover.de:8080 (Squid/2.2.PATCH2)
```

Erkennt ein Cache-Server seinen alphanumerischen Namen in einem Header, beendet er, unabhängig von seiner Konfiguration, die Weiterleitung des HTTP-Requests an weitere Neighbors und sendet den HTTP-Request direkt an den originalen WWW-Server. Durch den Aufruf der Methode `TRACE` in HTTP-Requests (s. Kapitel 2.1.3.2) und die nachfolgende Auswertung der General-Header `Via:` in den HTTP-Responses können generell Pfade in Cache-Verbünden untersucht werden.

2.3.2 Bildung von Cache-Verbünden

Welche benachbarten Cache-Server befragt und dabei als Parent oder Sibling genutzt werden sollen, wird auf den lokalen Cache-Servern konfiguriert. Um die Verteilung der HTTP-Requests an die Neighbors genauer lenken zu können, lassen sich in der Konfiguration zusätzliche Parameter angeben. Zu diesen Parametern gehört die Angabe von Toplevel Domains, so dass z. B. ein Neighbor nur nach Objekten auf WWW-Servern der Domain `com`, ein anderer nur nach Domains `de` und `edu` befragt werden kann. Weiterhin ist eine Gewichtung möglich, um ausgewählte Neighbors bevorzugt abzufragen.

Neben fest vorgegebenen Parametern können auch Informationen aus der Inter-Cache-Kommunikation, wie z. B. die Antwortzeiten von Neighbors, für eine dynamische Lenkung der HTTP-Requests ausgewertet werden. Hierdurch wird die aktuelle Lastsituation auf den Neighbors und in den Netzsegmenten berücksichtigt.

Die Unterscheidung zwischen Parent und Sibling ist von zentraler Bedeutung bei der Bildung von Cache-Verbünden. Zu beachten sind dabei folgende Aspekte:

- Ein Sibling wird nur dann von seinen Klienten befragt, wenn er die geforderten Objekte bereits gespeichert hat. Daraus folgt, dass sich der Speicher eines leeren Cache-Servers, der von seinen Klienten nur als Sibling genutzt wird, nicht füllen kann. Damit neue Objekte auf einem Cache-Server gespeichert werden, muss er von mindestens einem Klienten als Parent genutzt werden. Ein analoge Betrachtung führt zu folgender Feststellung:

  - Cache-Server konsumieren von ihren Siblings vorhandene Objekte.
  - Cache-Server produzieren auf ihren Parents neue Objekte.
- Die Parameter zur Nutzung übergeordneter Cache-Server werden auf den lokalen Cache-Servern konfiguriert. Der Betreiber der übergeordneten Cache-Server hat keine direkte Kontrolle über die Konfiguration der lokalen Cache-Server. Erst die Auswertung der Logdateien auf den zentralen Cache-Servern ergibt Aufschluss darüber, wie die Nutzung durch die lokalen Cache-Server erfolgt.
- Dem Betrieb eines übergeordneten Cache-Verbundes liegt in der Regel eine Konzeption hinsichtlich optimierter Verkehrslenkung oder Lastverteilung zugrunde. Wird eine solche Konzeption festgelegt, muss der Betreiber der übergeordneten Cache-Server die Administratoren der lokalen Cache-Server über entsprechende Hinweise zur Konfiguration informieren. Auf den übergeordneten Cache-Servern kann die Einhaltung der Konzeption nur durch ein ständiges Monitoring überwacht, jedoch nicht durch Einflussnahme auf die lokalen Cache-Server gesteuert werden [MSBV97].

2.3.3 Protokolle zur Inter-Cache-Kommunikation

In den folgenden Kapiteln werden die Protokolle *Internet Cache Protocol* (ICP), *Hypertext Caching Protocol* (HTCP), *Cache Digests* und *Cache Array Routing Protocol* (CARP) erläutert. Im Vordergrund stehen dabei die charakteristischen Merkmale, die für die Analyse und Modellierung der Protokolle in den nachfolgenden Kapiteln von Bedeutung sind. Die Betrachtung der Protokolle schließt jeweils mit einer Zusammenfassung verfügbarer Implementierungen.

2.3.3.1 ICP

Das Internet Cache Protocol (ICP) ist ein UDP-basiertes, verbindungsunabhängiges Protokoll, mit dem Informationen über einzelne Objekte zwischen Cache-Servern ausgetauscht werden. Jede ICP-Nachricht enthält neben einem festen Header die URL des geforderten Objekts. Nach jedem eingehenden HTTP-Request, der keinen unbestätigten Cache-Hit erzeugt, sendet ein lokaler Cache-Server ICP-Requests an seine Neighbors (Abbildung 2.14 links). Anhand definierter Opcodes im Header der ICP-Replies signalisieren die Neighbors, ob sie das geforderte Objekt gespeichert haben. Empfängt der lokale Cache-Server eine ICP-Reply mit einem ICP-Hit, fordert er das Objekt mit einem HTTP-Request von dem jeweiligen Neighbor ab (Abbildung 2.14 rechts).

ICP wurde als einfaches Protokoll konzipiert, um Übertragungszeiten und den Aufwand für die Verarbeitung gering zu halten. Mechanismen für eine gesicherte Übertragung von ICP-Nachrichten sind nicht vorgesehen, so dass bei einem Verlust einer ICP-Nachricht keine erneute Übertragung erfolgt. Stattdessen wird nach Ablauf eines Timeouts die Abfrage als gescheitert bewertet. Der Standardwert für den Timeout beträgt 2 Sekunden. Der tatsächlich verwendete Wert sollte sich jedoch an den

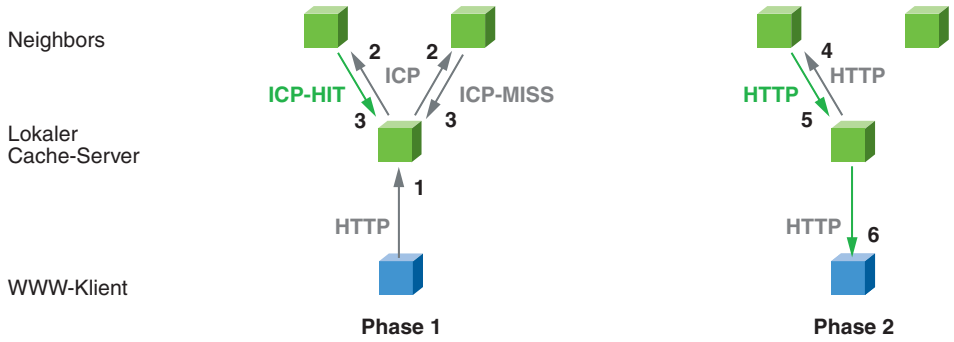


Abbildung 2.14: Inter-Cache-Kommunikation über ICP

Antwortzeiten der übergeordneten Cache-Server orientieren und kann in der Regel bei der Konfiguration der lokalen Cache-Server angegeben werden.

Unter Berücksichtigung der Protokolleigenschaften von ICP, wie der Unterscheidung von Siblings und Parents, der Gewichtung von Neighbors sowie möglichen Paketverlusten, ergeben sich bei der Auswertung der eingehenden ICP-Replies auf einem lokalen Cache-Server mehrere Möglichkeiten:

- Keine Antwort: Nach Ablauf der Timeouts sendet der lokale Cache-Server den HTTP-Request an einen Parent oder den originalen WWW-Server. Die Auswahl eines geeigneten Parents erfolgt nach dem Round-Robin-Verfahren oder richtet sich nach der Auswertung der Antwortzeiten vorangegangener ICP-Nachrichten.
- Nur ICP-Misses: Der HTTP-Request wird an den Parent mit der kürzesten Antwortzeit gerichtet. Gegebenenfalls wird eine Gewichtung der Parents bei der Auswahl berücksichtigt.
- Genau ein ICP-Hit: Der HTTP-Request wird an den Neighbor gerichtet, der den ICP-Hit signalisiert.
- Mehrere ICP-Hits: Der HTTP-Request wird an den Neighbor mit der kürzesten Antwortzeit gerichtet. Gegebenenfalls wird eine Gewichtung der Neighbors berücksichtigt.

Der ständige Austausch von ICP-Nachrichten zwischen einem lokalen Cache-Server und seinen Neighbors stellt implizit Informationen zur Verfügung, die eine Bewertung der aktuellen Lastsituation in einem Cache-Verbund ermöglichen. So wird der Ausfall oder die Überlastung eines übergeordneten Cache-Servers oder Netzsegments anhand fehlender oder stark verzögerter ICP-Replies von dem lokalen Cache-Servern sofort erkannt.

Aufbau von ICP-Nachrichten

Der Aufbau von ICP-Nachrichten in der Version 2 ist in RFC 2186 festgelegt [WeCl97a]. Neben dieser verbreiteten Version existiert eine proprietäre, nicht öf-

fentlich spezifizierte Version 3 des Protokolls [Wes01a]. Die Unterschiede zu der Version 2 sind jedoch marginal und für die folgenden Betrachtungen nicht von Bedeutung.

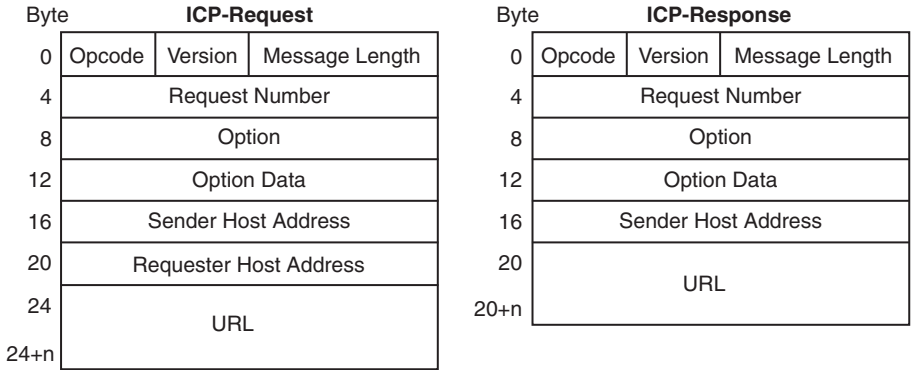


Abbildung 2.15: Aufbau von ICP-Nachrichten

Der in Abbildung 2.15 dargestellte Aufbau gilt für sämtliche ICP-Nachrichten. Die Größe einer ICP-Nachricht setzt sich aus dem festen Header sowie der Länge der URL zusammen. Im Unterschied zu ICP-Replies wird in der Payload des ICP-Requests zusätzlich die IP-Adresse des Klienten (*Requester Host Address*) angegeben, der den entsprechenden HTTP-Request an den Cache-Server gestellt hat. Hierdurch ist jeder ICP-Request 4 Byte länger als die entsprechende ICP-Reply. Die *Request Number* dient einem lokalen Cache-Server zusammen mit dem URL der Zuordnung von empfangenen ICP-Replies zu ausgesendeten ICP-Requests.

Das Feld *Opcode* enthält in einem ICP-Request den symbolischen Eintrag `ICP_OP_QUERY`. Die wichtigsten Einträge für Opcodes in ICP-Replies sind `ICP_OP_HIT` für einen ICP-Hit sowie `ICP_OP_MISS` für einen ICP-Miss. Durch den alternativen Opcode `ICP_OP_MISS_NOFETCH` signalisiert ein übergeordneter Cache-Server mit einem IPC-Miss zugleich die Forderung, dass das Objekt nicht mit einem nachfolgenden HTTP-Request abgerufen werden soll. Somit kann ein übergeordneter Cache-Server anzeigen, dass er für den Abruf dieses Objekts nicht als Parent zur Verfügung steht. Nach der Empfehlung für die Implementierung von ICP soll ein Cache-Server nur dann einen ICP-Hit signalisieren, wenn das Objekt noch für einen Zeitraum von mindestens 30 Sekunden als aktuell beurteilt werden kann [WeCl97b]. Diese Forderung zielt auf die Vermeidung falscher Cache-Hits (Kapitel 2.3.1).

Implementierung

Aufgrund der einfachen Gestaltung von ICP sind Implementierungen ohne höheren Aufwand möglich. Die Generierung und Auswertung der ICP-Nachrichten kann durch einfache Operation im Hauptspeicher erfolgen, sie stellen daher keine besonderen Leistungsanforderungen an einen Cache-Server dar [WeCl98].

Gegenwärtig wird ICP in der Version 2 von fast allen verfügbaren Cache-Servern unterstützt.

2.3.3.2 HTCP

Das in RFC 2756 spezifizierte Hypertext Caching Protocol (HTCP) stellt eine Weiterentwicklung von ICP dar [ViWe00]. Durch eine Erweiterung des Nachrichtenformats ermöglicht HTCP die Authentifizierung von Nachrichten sowie die Übertragung von Header-Informationen aus HTTP-Requests und HTTP-Responses. Da Cache-Server mit HTCP-Nachrichten Informationen über einzelne Objekte austauschen, entspricht der Ablauf der Kommunikation dem in ICP (vgl. Abbildung 2.14).

Als schwerwiegender Mangel von ICP wird die fehlende Authentifizierung von Nachrichten angesehen [WeCl97b]. Durch Abfangen und Manipulieren von ICP-Nachrichten in einem Cache-Verbund ist es theoretisch möglich, gefälschte Inhalte auf Cache-Servern und damit an die WWW-Klienten zu verteilen. In HTCP enthält jede Nachricht einem Hashwert nach HMAC-MD5 [KBC97] als Signatur, mit der der Empfänger die Authentizität prüfen kann. Um gleichzeitig mit verschiedenen Neighbors unterschiedlich signierte HTCP-Nachrichten austauschen zu können, muss ein Cache-Server mehrere geheime Schlüssel verwalten. Zur Identifizierung der verwendeten Schlüssel enthält jede HTCP-Nachricht neben der eigentlichen Signatur auch den Namen des geheimen Schlüssels. Der Austausch der geheimen Schlüssel, die zur Bildung und Prüfung der Signatur sowohl dem Sender als dem Empfänger bekannt sein müssen, erfolgt außerhalb von HTCP.

Bei der Auswertung von ICP-Replies zeigt sich ein weiterer Mangel von ICP. Bei falscher Abschätzung der Aktualität ist es möglich, dass Neighbors ICP-Hits signalisieren, obwohl sie veraltete Objekte vorhalten. Dem lokalen Cache-Server fehlen jegliche Informationen, um die Aktualität der angebotenen Objekte zu beurteilen. In HTCP werden daher mit den HTCP-Responses die HTTP-Header der Objekte an die lokalen Cache-Server übermittelt. Mit diesen Informationen erhalten untergeordnete Cache-Server ein weiteres Entscheidungskriterium für die Wahl des geeigneten Neighbors.

Implementierung

Ein Zugriff auf die Festplatten, um die für die Generierung einer HTCP-Response notwendigen HTTP-Header eines Objekts aus dem Dateisystem auszulesen, würde die Antwortzeiten von HTCP wesentlich verlängern. Für eine verzögerungsfreie Übertragung ist es daher notwendig, dass ein Cache-Server die HTTP-Header der Objekte im Hauptspeicher vorhält. Der bisherige Speicherbedarf eines Cache-Servers wird wesentlich durch die 16 Byte breiten MD5-Hashwerte der URLs für jedes gespeicherte Objekt bestimmt. Das Volumen der HTTP-Header übersteigt diesen Bedarf um ein Vielfaches (vgl. Anhang A.3), so dass eine effiziente Implementierung von HTCP einen großen verfügbaren Hauptspeicher voraussetzt. Weiterhin be-

nötigt HTCP im Vergleich zu ICP durch die Bildung und Prüfung der Signaturen in jeder Nachricht eine höhere Prozessorleistung.

Neben einer experimentellen Implementierung in der Cache-Software Squid (s. Anhang A.3) sind bisher keine Cache-Server bekannt, die HTCP unterstützen.

2.3.3.3 Cache Digests

Cache Digests stellen eine Alternative zu Protokollen wie ICP oder HTCP dar, die mit jeder Nachricht Informationen über einzelne Objekte austauschen. Ein Cache Digest enthält ein Verzeichnis aller Objekte, die momentan auf einem Cache-Server vorgehalten werden. Der Abruf der Cache Digests erfolgt durch die untergeordneten Cache-Server zu festgelegten Zeitpunkten.

Der erste Ansatz für die Erstellung und Übertragung eines Verzeichnisses aller auf einem Cache-Server gespeicherten Objekte wurde mit *Summary Cache* vorgestellt [FCAB00]. Summary Cache war als Erweiterung von ICP konzipiert, indem die Verzeichnisse mit ICP-Nachrichten zwischen Cache-Servern ausgetauscht werden sollten. Es stellte sich jedoch heraus, dass ICP für die Übertragung umfangreicher Objekte nicht geeignet ist [KCYS99]. Unter der Bezeichnung *Cache Digests* wurden die Ansätze von Summary Cache in der Cache-Software Squid implementiert [RoWe98]. Gegenüber Summary Cache wird ein Cache Digest als eigenständiges WWW-Objekt betrachtet, so dass die Übertragung über HTTP möglich ist. Der Abruf der Cache Digests von einem Cache-Server erfolgt über definierte URLs.

Die Herausforderung von Summary Cache und Cache Digests liegt in der Generierung eines komprimierten Verzeichnisses aller gespeicherten URLs. Die Übertragung von Verzeichnissen, die die URLs im Klartext oder als MD5-Hashwerte enthalten, scheidet aus zwei Gründen aus:

- Da ein lokaler Cache-Server die Cache Digests sämtlicher Neighbors im Hauptspeicher vorhält, muss das Datenvolumen eines einzelnen Cache Digests gering gehalten werden.
- Für jeden Cache-Miss muss ein lokaler Cache-Server das geforderte Objekt in den Cache Digests sämtlicher Neighbors suchen. Die Dauer der Suche steigt mit zunehmendem Umfang der Cache Digests.

Als geeignete Form für ein komprimiertes Verzeichnis werden in Summary Cache die *Bloom Filter* eingeführt [Blo70]. In diesem Verfahren wird das Verzeichnis der N auf einem Cache-Server gespeicherten Objekte mit den URLs $U = \{u_1,\ldots,u_N\}$ durch einen Vektor $\vec{v}$ repräsentiert. Der Vektor enthält M Elemente, die im Ausgangszustand auf 0 gesetzt sind. Da die Elemente nur die Zustände 0 und 1 annehmen sollen, kann jedes Element des Vektors durch ein Bit dargestellt werden. Für die Abbildung der URLs U auf $\vec{v}$ werden K voneinander unabhängige Hashfunktionen $h_1,\ldots,h_K$ definiert. Jede Hashfunktion erzeugt einen Wert im Bereich $\{1,\ldots,M\}$. Ein Aufruf der Hashfunktionen mit dem URL u_i erzeugt die Werte

$h_1(u_i),\ldots,h_K(u_i)$. Diese Werte kennzeichnen die Positionen der Elemente in $\vec{v}$, die auf 1 gesetzt werden. Um zu prüfen, ob ein URL u_j in $\vec{v}$ enthalten ist, müssen die Elemente an den Positionen $h_1(u_j),\ldots,h_K(u_j)$ auf 1 gesetzt sein. Ist eines der Elemente auf 0 gesetzt, kann u_j nicht in $\vec{v}$ verzeichnet sein.

Die Abbildung von U auf $\vec{v}$ ist nicht eindeutig umkehrbar. Verschiedene URLs u_i und u_j können dieselben Hashwerte $h_1(u_i)=h_1(u_j),\ldots,h_K(u_i)=h_K(u_j)$ ergeben. Daraus folgt, dass die Suche nach Einträgen in $\vec{v}$ falsche Treffer (*false drops*) ergeben kann. Die Wahrscheinlichkeit für falsche Treffer P_{false} lässt sich bei Bloom Filtern in Abhängigkeit der Größen K, M und N exakt angeben:

$$P_{\text{false}} = \left(1-\left(1-\frac{1}{M}\right)^{KN}\right)^K \approx \left(1-\mathrm{e}^{-\frac{KN}{M}}\right)^K \tag{2.3}$$

Es ist zu beachten, dass mit der Verwendung von Cache Digests gegenüber ICP das Auftreten falscher Cache-Hits steigt. Der Anstieg ist durch den in Kapitel 2.3.1 erläuterten Wert T_Δ begründet. Bei ICP entspricht dieser Wert der Umlaufzeit einer ICP-Nachricht von wenigen Millisekunden, bei Cache Digests maximal dem Zeitraum zwischen dem Abruf von zwei Cache Digests, der in der Regel eine Stunde beträgt.

Implementierung

Zur effizienten Implementierung eines Bloom Filters müssen die Parameter K, M und N geeignet bestimmt werden. Die Größenordnung von N ist durch das Volumen des Cache-Servers und die mittlere Größe der Objekte im Speicher vorgegeben. Für eine schnelle Auswertung des Bloom Filters sollte die Anzahl K der Hashfunktionen gering sein. Da sämtliche Vektoren der Neighbors im Hauptspeicher vorgehalten werden müssen, sollte deren Größe von M Bits ebenfalls möglichst gering sein.

Für die Wahl geeigneter Parameter werden in Summary Cache bereits mehrere Angaben vorgeschlagen. Die Abschätzung in Gleichung 2.3 für die Wahrscheinlichkeit P_{false} erreicht ein Minimum bei $K=\ln 2 \cdot M/N$. Da der Vektor $\vec{v}$ eine Folge von Bits repräsentiert, kann der Faktor M/N als das Verhältnis von Bits pro URL betrachtet werden. Die Wahrscheinlichkeit P_{false} lässt sich so in Abhängigkeit des Faktors M/N für verschiedene K darstellen (Abbildung 2.16).

In der Cache-Software Squid werden Cache Digests mit 5 Bits pro URL generiert [HRW98]. Zusammen mit der Anzahl N der gespeicherten Objekte ergibt sich daraus die Größe M des Cache Digests. Jedes Objekt wird in dem Cache Digest durch $K=4$ Hashwerte repräsentiert. Hierfür wird der MD5-Hashwert eines URLs in vier gleiche Teile zu je 32 Bits aufgeteilt, die jeweils durch eine Modulo-Operation mit M die Position in M angeben, die auf den Wert 1 gesetzt wird. Da die MD5-Hash-

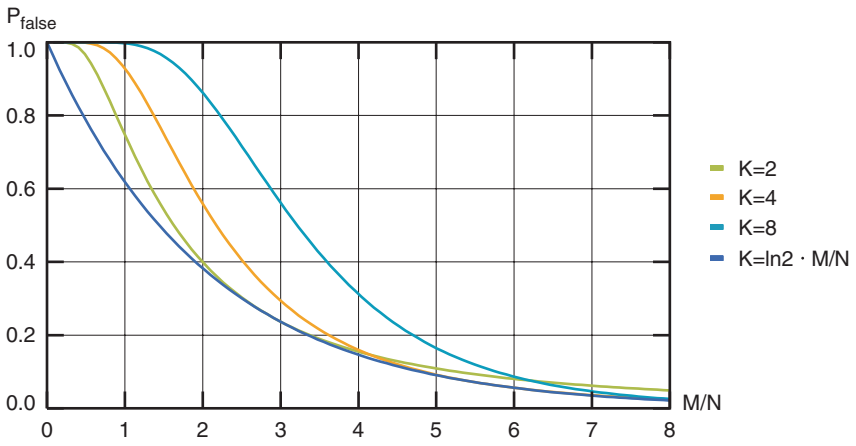


Abbildung 2.16: Auswahl Parameter für Bloom Filter

werte bereits für das eigentliche Inhaltsverzeichnis der Objekte auf dem Cache-Server generiert werden müssen, entsteht durch die Bildung und Auswertung der Cache Digests nur ein geringer zusätzlicher Rechenaufwand. Aus den genannten Werten für K, M und N ergibt sich in Squid eine Wahrscheinlichkeit P_{false} von 9,2%.

Einem Cache Digest wird bei der Übertragung ein Header vorangestellt, der unter anderem die Angabe von K, M und N enthält. Diese Angaben sind eine notwendige Voraussetzung für die Auswertung des Cache Digests auf dem Empfänger.

Die Implementierung von Cache Digests in kommerziellen Cache-Servern ist derzeit nicht zu erwarten, da die University of Wisconsin ein Patent auf die Ergebnisse von Summary Cache beantragt hat [CMT01].

2.3.3.4 Vergleichende Betrachtung von Inter-Cache-Protokollen

In Abbildung 2.17 werden die betrachteten Verfahren zur Inter-Cache-Kommunikation in einem Protokoll-Stack dargestellt. Die grau hinterlegten Felder deuten an, dass sowohl ICP als auch HTCP prinzipiell auch für die Übertragung über TCP geeignet sind. Aufgrund der erforderlichen schnellen Bearbeitung und der tolerierbaren Verluste nutzen bestehende Implementierungen jedoch ausschließlich UDP [Wan99].

Im Gegensatz zu ICP und HTCP erfordern Cache-Digests eine gesicherte Übertragung. Da HTTP als Übertragungsprotokoll auf Cache-Servern bereits implementiert ist, bietet sich dessen unmittelbare Nutzung für die Übertragung an. Der hierdurch entstehende Overhead ist nicht relevant, da Cache Digests in der Regel lediglich stündlich abgerufen werden.

Da die Häufigkeit und das Datenvolumen, das bei der Übertragung von Cache Digests anfällt, ausschließlich durch die Konfiguration und die Größe der Cache-Server bestimmt wird, lässt sich das entsprechende Verkehrsaufkommen annähernd ex-

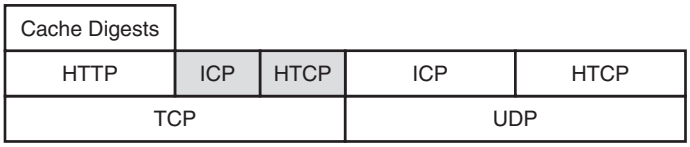


Abbildung 2.17: Einordnung Inter-Cache-Protokolle

akt angeben. Im Gegensatz dazu ist das Aufkommen an ICP- und HTCP-Verkehr von der jeweiligen Lastsituation abhängig. Eine vergleichende Gegenüberstellung unter der Zielsetzung, das geeignete Protokoll für eine vorgegebene Konfiguration festzulegen, kann daher nur unter vereinfachenden Annahmen durchgeführt werden.

Im Folgenden werden zunächst die Datenvolumen ermittelt, die sich aus den charakteristischen Eigenschaften der Protokolle ergeben. Ausgehend davon werden die mittleren Raten F_{ICP} und F_{HTCP} für den Austausch von ICP- bzw. HTCP-Nachrichten in Abhängigkeit der Größe von Cache Digests $S_{\mathrm{Cache\ Digests}}$ dargestellt. Als Ergebnis lassen sich die maximalen mittleren Raten von ICP und HTTP angeben, oberhalb denen der Einsatz von Cache Digests eine Einsparung des übertragenen Datenvolumens erbringt.

Für die Berechnung wird die Häufigkeit $F_{\mathrm{Cache\ Digests}}$, mit der Cache Digests abgerufen werden, entsprechend den Angaben in Kapitel 2.3.3.3 auf 1 Stunde gesetzt. Für den Austausch von ICP-Request und -Reply ergibt sich nach Tabelle 3.15 eine mittlere Größe S_{ICP} von $2 \cdot 76$ Byte. Nach Anhang A.3 ergibt sich für HTCP eine mittlere Größe S_{HTCP} von 589+110 Byte. Daraus lassen sich die mittleren Datenraten R mit folgenden Gleichungen bestimmen:

$$\begin{aligned} R_{\mathrm{ICP}} &= S_{\mathrm{ICP}} \cdot F_{\mathrm{ICP}} = 152\ \mathrm{Byte} \cdot F_{\mathrm{ICP}} \\ R_{\mathrm{HTCP}} &= S_{\mathrm{HTCP}} \cdot F_{\mathrm{HTCP}} = 699\ \mathrm{Byte} \cdot F_{\mathrm{HTCP}} \\ R_{\mathrm{Cache\ Digests}} &= S_{\mathrm{Cache\ Digests}} \cdot F_{\mathrm{Cache\ Digests}} = \frac{S_{\mathrm{Cache\ Digests}}}{3.600\ \mathrm{s}} \end{aligned} \tag{2.4}$$

Durch Gleichsetzung $R_{\mathrm{Cache\ Digests}} = R_{\mathrm{ICP}}$ und $R_{\mathrm{Cache\ Digests}} = R_{\mathrm{HTCP}}$ ergeben sich die in Abbildung 2.18 dargestellten Funktionen für F_{ICP} und F_{HTCP} in Abhängigkeit von $S_{\mathrm{Cache\ Digests}}$.

Das Diagramm zeigt, dass der Einsatz von Cache Digests mit einem Volumen von 2 MByte bereits dann sinnvoll ist, wenn die die mittleren Raten von ICP 3,8 Requests pro Sekunde oder von HTCP 0,8 Requests pro Sekunde übersteigen.

2.3.3.5 CARP

Das Cache Array Routing Protocol (CARP) stellt kein Inter-Cache-Protokoll im eigentlichen Sinne dar, da keine Kommunikation zwischen den Cache-Servern stattfindet. Da CARP dennoch zusammen mit WCCP in zahlreichen Veröffentlichungen

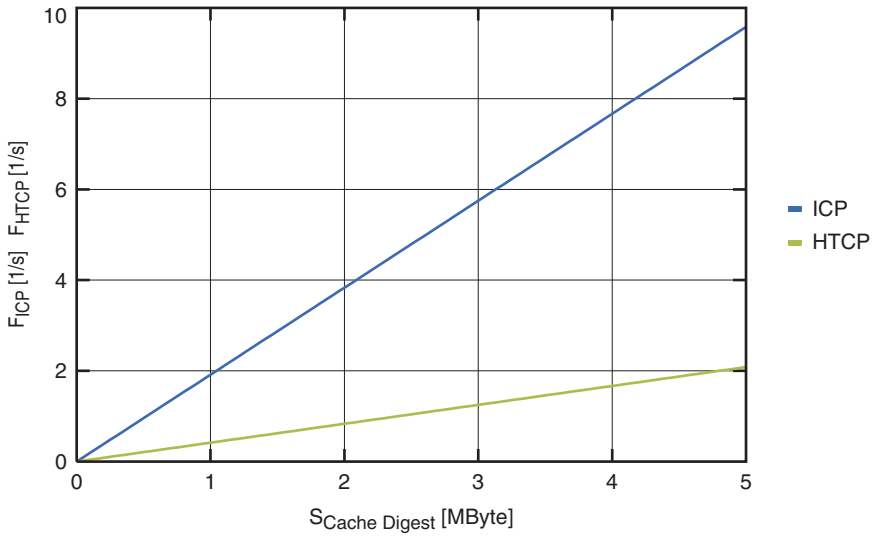


Abbildung 2.18: Vergleich ICP und HTCP gegenüber Cache Digests

den hier betrachteten Inter-Cache-Protokollen gleichgestellt wird, werden im Folgenden kurz die wesentlichen Merkmale aufgeführt.

Mit CARP wird ein Algorithmus definiert, nach dem ein lokaler Cache-Server HTTP-Requests als Funktion der URLs auf seine Neighbors verteilt [VaRo98]. Diese determinierte Verkehrslenkung wurde mit folgenden Zielsetzungen konzipiert:

- Verzicht auf Inter-Cache-Kommunikation, dadurch keine Verzögerung durch Übertragungszeiten und keine zusätzliche Belastung der Netze.
- Optimale Verteilung der Objekte im Cache-Verbund, d. h. auf der Gesamtheit der übergeordneten Cache-Server ist jedes Objekt nur einmal vorhanden. Die Forderung einer optimalen Verteilung wird in [VaRo98] wie folgt formuliert: Einem lokalen Cache-Server sind K Neighbors zugewiesen, O_i bezeichnet die Menge der Objekte auf dem Neighbor i. Daraus ergibt sich für eine optimale Verteilung der Objekte die Forderung $O_i \cap O_j = \varnothing$ für alle $i,j \in K,\ i \neq j$.
- Die Verteilung der HTTP-Requests auf die übergeordneten Cache-Server kann in Abhängigkeit verschiedener Parameter gewichtet werden.
- Eine Veränderung der Anzahl Neighbors darf nur zu einer geringen Veränderung bei der Verteilung der HTTP-Requests führen.
- Die Abbildung der URLs auf die übergeordneten Cache-Server erfolgt über Hashwerte.

Bereits aus der Konzeption von CARP geht hervor, dass dieser Ansatz nur für eng gekoppelte Cache-Server bzw. so genannte Cache-Cluster geeignet ist, in denen eine hohe Verfügbarkeit gewährleistet werden kann. Die fehlende Inter-Cache-Kommunikation erfordert unter anderem, dass die Verfügbarkeit der Neighbors durch zusätzliche Maßnahmen überwacht wird. Weiterhin führen Störungen auf Netzseg-

menten oder benachbarten Cache-Server dazu, dass sich das Routing der HTTP-Requests innerhalb eines CARP-basierten Verbunde ändert. Auch wenn durch den Einsatz intelligenter Hashing-Verfahren die Auswirkungen häufiger Änderungen reduziert werden, lässt sich CARP nicht für die Kopplung von Cache-Servern über Weitverkehrsnetze einsetzen. Für die weiteren Betrachtungen in dieser Arbeit sind CARP und WCCP nicht relevant.

Implementierung

Da in CARP einfache Verfahren für die Generierung der Hashwerte verwendet werden, sind die Anforderungen an Prozessor und Hauptspeicher gering [Ros97]. CARP ist in der Cache-Software Squid sowie in dem Microsoft Proxy Server [MS97] implementiert.

2.3.3.6 Alternative Ansätze

Neben den aufgeführten Protokollen und Konzepten für die Bildung von Cache-Verbünden finden sich in der Literatur zahlreiche weiterführende Untersuchungen und Vorschläge. Die dort untersuchten Aspekte lassen sich in folgende Bereiche einteilen:

ICP über Multicast
: Anstatt einzelne ICP-Requests dediziert an jeden Neighbor zu senden, können Cache-Server in einer Multicast-Gruppe organisiert werden. Die Inter-Cache-Kommunikation erfolgt mit ICP über Multicast, entsprechende Erweiterungen wurden in der Cache-Software Squid bereits in der Version 1.1.5 implementiert.

Proaktive Aktualisierung von Objekten, Pushing
: Sich häufig ändernde Objekte werden über unicast- [HMY97] oder multicast-basierte Strukturen [RSB01] [ToHu98] [LiCh99] in einem Cache-Verbund aktualisiert.

Dynamische Bildung von Cache-Verbünden
: Unter Verwendung des multicast-basierten *Cache Group Management Protocols* (CGMP) können sich einzelne Cache-Server selbstständig zu überlappenden Cache-Verbünden gruppieren. Die Bildung der Gruppen orientiert sich an den auf den Cache-Servern gespeicherten Objekten [ZMN+98].

Globale Verzeichnisse der gespeicherten Objekte
: Einzelne Cache-Server legen die Metainformationen der gespeicherten Objekte in globalen Verzeichnissen auf dedizierten Servern ab. Durch kontinuierliche Aktualisierung der Informationen können Cache-Server direkt auf die Nachbarn zugreifen, auf denen ein gefordertes Objekt gespeichert ist [PoHa97] [DJD99].

In einem weiterführenden Ansatz werden die Metainformationen geeignet partitioniert und direkt von den Cache-Servern vorgehalten [TDVK98].

Zentrale Verteilung von HTTP-Requests
Auf einem zentralen Server werden alle in einem Cache-Verbund gespeicherten Objekte verzeichnet. Der zentrale Server nimmt die HTTP-Requests der Klienten (WWW-Server oder lokale Cache-Server) entgegen und verteilt sie an die entsprechenden Cache-Server [GRC97].

Lenkung der HTTP-Requests in Abhängigkeit von Routing-Informationen
In größeren Kernnetzen sind die Übergangspunkte in Netze anderer Betreiber auf mehrere Standorte verteilt. Werden mehrere zentrale Cache-Server auf diese Standorte verteilt, können Routing-Informationen für die Auswahl des geeigneten übergeordneten Cache-Servers ausgewertet werden. Die Lenkung der HTTP-Requests an die zentralen Cache-Server erfolgt auf den „natürlichen“ Routen zu den originalen WWW-Servern, wodurch überflüssige Belastungen im Kernnetz und dadurch höhere Verzögerungszeiten vermieden werden können [VGP98].

Die aufgeführten Untersuchungen sind für den praktischen Betrieb von Cache-Verbünden bisher nicht relevant. Als einzige dokumentierte Ausnahme wurde bisher ICP über Multicast in einem größeren Cache-Verbund erfolgreich eingesetzt [NeHe97]. Aufgrund der mangelnden Stabilität multicast-basierter Netze wird jedoch generell von diesem Verfahren abgeraten [Wes01b].

Im Gegensatz zu verteilten Speicherarchitekturen konnten bisher keine Protokolle für Cache-Verbünde etabliert werden, die die Kohärenz der Objekte auf den Cache-Servern sicherstellen. Neben den in Kapitel 2.2.2.5 aufgeführten Verfahren für die Gewährleistung einer starken Konsistenz, die sich auch in Cache-Verbünden anwenden lassen, sind in der Literatur zwar verschiedene Ansätze zu finden [KrWi97] [DAPJ99] [CEB+00] [LCD00]. Bisher liegen jedoch keine stabilen Implementierungen oder Untersuchungen aus dem praktischen Betrieb vor. Alternative Vorschläge basieren auf Multicast-Strukturen, in denen die Aktualisierung und Verteilung der Objekte durch die WWW-Server initiiert werden [YBS99] [RBR01].

Kapitel 3

Analyse

3.1 Der DFN-Cache-Verbund

Im deutschen Wissenschaftsnetz wurde Anfang 1996 der erste Verbund lokaler Cache-Server zwischen den Universitäten Bochum, Bonn und Frankfurt gebildet. In den folgenden Monaten erweiterten zahlreiche Cache-Server anderer Einrichtungen diesen Cache-Verbund, so dass bald ein unkoordiniertes und unstrukturiertes Netz von Cache-Servern im Wissenschaftsnetz vorhanden war. Aus dieser Situation heraus wurde das Projekt *Konzeption einer Cache-Server-Infrastruktur im Wissenschaftsnetz* gestartet. Das vom DFN-Verein aus Mitteln des BMBF geförderte Projekt wurde vom 1. August 1996 bis zum 31. Dezember 2000 am Lehrgebiet Rechnernetze und Verteilte Systeme der Universität Hannover durchgeführt.

Das übergeordnete Ziel des Projektes bestand zunächst in der Klärung der Grundlagen für einen zentralen Cache-Service im Wissenschaftsnetz. Neben der Entflechtung und Koordinierung des oben dargestellten losen Verbundes lokaler Cache-Server sollte durch den Aufbau eines so genannten DFN-Cache-Verbundes auch der WWW-Verkehr auf den stark belasteten Übertragungsstrecken zwischen den USA und dem Wissenschaftsnetz gezielt reduziert werden. Da seitens der nutzenden Einrichtungen dringender Bedarf an einer übergeordneten Cache-Server-Infrastruktur im Wissenschaftsnetz bestand, wurden parallel zu diesen Überlegungen zehn DFN-Cache-Server installiert. Die DFN-Cache-Server wurden direkt an die zentralen Router im Kernnetz des damaligen Breitband-Wissenschaftsnetzes (B-WiN) angeschlossen und bildeten ab Januar 1997 die oberste Hierarchiestufe des DFN-Cache-Verbundes. Ende Februar 2000 wurden die zehn DFN-Cache-Server durch vier leistungsfähigere Systeme ersetzt. Zeitweise waren bis zu 200 lokale Cache-Server von Hochschulen und Forschungseinrichtungen an den DFN-Cache-Verbund angeschlossen.

Um den während der Projektlaufzeit kontinuierlich steigenden Anforderungen gerecht zu werden, musste die Konfiguration des gesamten DFN-Cache-Verbundes mehrfach angepasst werden. Anhand der unterschiedlichen Umgebungen konnten über einen Zeitraum von vier Jahren Informationen über die Verteilung von Ver-

kehrsflüssen innerhalb eines Cache-Verbundes gesammelt und Beobachtungen über die allgemeine Entwicklung des WWW-Verkehrs durchgeführt werden. Die Ergebnisse der Verkehrsanalysen werden in diesem Kapitel vorgestellt. Den Analysen wird ein Überblick über den Verlauf des Projektes sowie die den DFN-Cache-Verbünden zugrunde liegende Ausstattung und Konzeption vorangestellt. Die darin getroffenen qualitativen Aussagen über das Verkehrsaufkommen und das Lastverhalten der DFN-Cache-Server werden mit den Analysen quantitativ belegt.

3.1.1 Ausstattung der DFN-Cache-Server

Während des gesamten Projektverlaufs wurden ausschließlich UNIX-basierte Cache-Server mit der Cache-Software Squid betrieben. Hardware-basierte Cache-Server konnten aufgrund der hohen Investitionskosten nicht eingesetzt werden. Die Einschränkungen, die sich aus dem Einsatz software-basierter Cache-Server ergeben, wurden bereits in Kapitel 2.2.2.4 erläutert. Maßnahmen auf den DFN-Cache-Servern zur Optimierung der Cache-Prozesse sowie des zugrunde liegenden Betriebssystems wurden mit den jeweiligen Abschlussberichten der Projektteile dargelegt [GVP98a] [GVP01] und sind nicht Inhalt dieser Arbeit.

	alter DFN-Cache-Verbund	**neuer DFN-Cache-Verbund**
Anzahl Server	10	2
Bezeichnung	Sun Ultra Enterprise 2	Sun Ultra Enterprise 450
Prozessoren	2 x 167 MHz	4 x 400 MHz
Hauptspeicher	256 MByte	2 GByte
Plattenplatz	15 GByte	180 GByte
Netzinterface	1 x Fast Ethernet	2 x Fast Ethernet
Betriebssystem	Solaris 2.5.1 / 2.7	Solaris 2.7

Tabelle 3.1: Hardware der DFN-Cache-Server

Tabelle 3.1 enthält eine Gegenüberstellung der Hardware von alten und neuen DFN-Cache-Servern. Die Hardware der alten DFN-Cache-Server erschien zunächst ausreichend dimensioniert, um einen leistungsfähigen Cache-Verbund dauerhaft zu betreiben. Allerdings konnten die zehn Cache-Server nicht ausschließlich für das Caching genutzt werden, sondern wurden auch für weitere Dienste im B-WiN, wie die Sammlung von IP-Statistiken und die Verteilung von News, eingesetzt. Aufgrund der steigenden Anforderungen stellten die alten DFN-Cache-Server mit fortlaufender Projektdauer keine optimale Plattform mehr dar.

Mit der Installation von zwei neuen, ausschließlich für das Caching vorgesehenen DFN-Cache-Servern am Standort Hannover konnten die alten DFN-Cache-Server im Frühjahr 2000 außer Betrieb genommen werden. Die neuen DFN-Cache-Server

waren so dimensioniert, dass jeweils zwei voneinander unabhängige Cache-Prozesse parallel ausgeführt werden konnten. Insgesamt zeigt ein Vergleich der verfügbaren Hardware von alten und neuen DFN-Cache-Servern (Tabelle 3.1) sowie die Gegenüberstellung der tatsächlich für das Caching genutzten Ressourcen (Tabelle 3.2, Stand Februar 2000 bzw. November 2000) deutliche Unterschiede. Während die alten DFN-Cache-Server gegen Ende des Projekts dauerhaft an ihrer Lastgrenze betrieben wurden, waren auf den neuen DFN-Cache-Servern auch zu Spitzenzeiten Leistungsreserven vorhanden.

	alter DFN-Cache-Verbund	**neuer DFN-Cache-Verbund**
max. CPUs pro Cache-Prozess	1	2
Anbindung an B-WiN über zentrale Router	Ethernet	Fast Ethernet
max. Plattenplatz pro Cache-Prozess	4 GByte	56 GByte
ges. Volumen auf allen DFN-Cache-Servern	40 GByte	224 GByte
max. zulässiges Volumen je Objekt	32 MByte	96 MByte
daraus resultierend		
ges. Objekte auf allen DFN-Cache-Servern	< 2 Millionen	> 10 Millionen
max. Verweildauer der Objekte im Cache	20 Stunden	85 Stunden

Tabelle 3.2: Parameter von altem und neuem DFN-Cache-Verbund

Die Anbindung der alten DFN-Cache-Server an das B-WiN konnte nicht mit Fast Ethernet erfolgen, da die zugehörigen zentralen Router nur über Ethernet-Interfaces verfügten. Die neuen DFN-Cache-Server konnten dagegen auch über Fast Ethernet angeschlossen werden, wobei die Kommunikation der DFN-Cache-Server untereinander direkt über ein zusätzliches Fast Ethernet-Interface erfolgte (s. Abbildung 3.5).

Aufgrund des wesentlich höheren Plattenvolumens wurde im neuen DFN-Cache-Verbund die maximal zulässige Größe der gespeicherten Objekte auf 96 MByte erhöht. Somit konnten selbst größere Objekte gespeichert und über einen längeren Zeitraum zur Verfügung gestellt werden.

Ein deutlicher Hinweis auf die bessere Ausstattung der neuen DFN-Cache-Server konte aus der maximalen Verweildauer der Objekte abgeleitet werden, die eine wesentliche Orientierung bei der Dimensionierung von Cache-Servern darstellt. Die tatsächliche maximale Verweildauer der Objekte sollte die konfigurierte Verweil-

dauer, die die maximale Zeitspanne bis zu einer Überprüfung der Aktualität auf dem originalen WWW-Server angibt (s. Kapitel 2.2.2.6), nicht unterschreiten [Wes01a]. Die geringe tatsächliche Verweildauer von unter einem Tag auf den alten DFN-Cache-Servern deutete auf zu gering dimensionierten Speicher hin. Mit dem mehr als fünffachen Volumen auf den neuen DFN-Cache-Servern stieg die maximale Verweildauer deutlich an.

3.1.2 Konzeption

Bereits vor der Inbetriebnahme der zehn DFN-Cache-Server im Januar 1997 musste eine Konzeption zur Last- und Verkehrsverteilung sowie zur Nutzung durch die lokalen Cache-Server festgelegt werden. Da bekannt war, dass das Kernnetz des B-WiN stetigen Änderungen unterworfen sein wird (siehe Abbildung 3.1), durfte sich die Konzeption des DFN-Cache-Verbundes nicht zu stark an dessen Topologie orientieren.

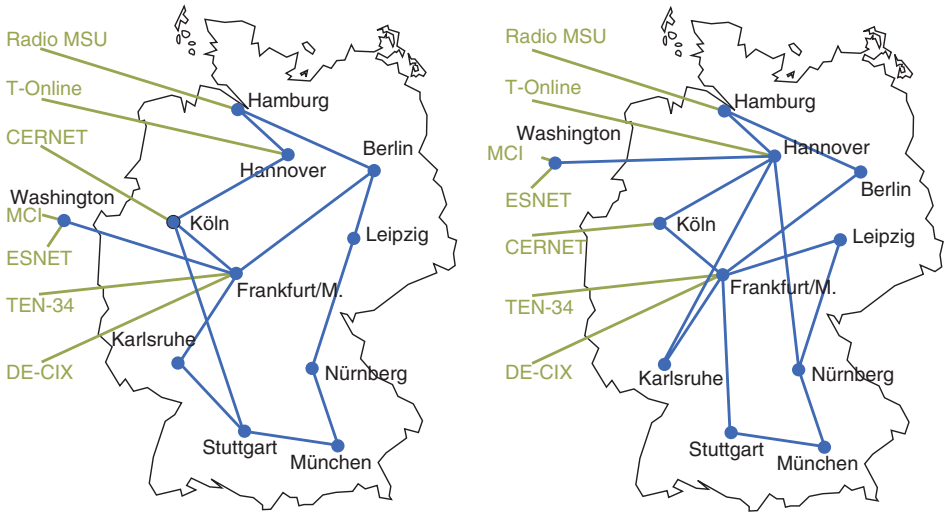


Abbildung 3.1: Kernnetz im Breitband-Wissenschaftsnetz

In einem ersten Schritt wurde die Konzeption des im amerikanischen Forschungsnetz NSFNET betriebenen Cache-Verbundes übernommen [IRC02]. Anhand der gesammelten Betriebserfahrungen und Verkehrsstatistiken wurde ein halbes Jahr nach Inbetriebnahme des DFN-Cache-Verbundes eine eigenständige Konzeption entworfen und umgesetzt. Diese Konzeption wurde bis zur Beendigung des alten DFN-Cache-Verbundes beibehalten und zum Teil auch für den neuen DFN-Cache-Verbund übernommen.

3.1.2.1 Alter DFN-Cache-Verbund – ursprüngliche Konzeption

Abgesehen von einem Projekt des National Laboratory for Applied Network Research (NLANR), in dem ein übergeordneter Cache-Verbund für das amerikanische Forschungsnetz NSFNET betrieben wurde, existierte zum Zeitpunkt des Projektbeginns kein Cache-Verbund vergleichbarer Größe. Da innerhalb des DFN keine gesicherten Informationen über den WWW-Verkehr im Wissenschaftsnetz vorlagen, orientierte sich die erste Konzeption des DFN-Cache-Verbundes an dem Cache-Verbund des NLANR. In einem ersten Entwurf wurden Zuständigkeiten für Objekte von WWW-Servern aus 50 verschiedenen Toplevel Domains auf die DFN-Cache-Server verteilt. Die in Tabelle 3.3 dargestellte Verteilung von 50 Toplevel Domains erfolgte nach dem geschätzten Verkehrsaufkommen und unter geographischen Gesichtspunkten.

Toplevel Domain	DFN-Cache-Server
cz hu pl ru si sk su tr	Berlin
com	Frankfurt, Köln
cn id in jp kr my sg th tw vn	Hamburg
dk fi gr is no se sp pt	Hannover
at be lu ch fr it nl	Karlsruhe
de	Leipzig
net	München
edu gov int mil org us	Nürnberg
au br ca ie mx nz uk za	Stuttgart

Tabelle 3.3: Verteilung von 50 Toplevel Domains auf DFN-Cache-Server

Für die Anbindung der lokalen Cache-Server und die Lenkung der Requests innerhalb des DFN-Cache-Verbundes wurde folgende Konfiguration vorgesehen:

- Jedem lokalen Cache-Server wurden zwei benachbarte DFN-Cache-Server als Parents zugeordnet. Eine Lenkung der Requests in Abhängigkeit der Toplevel Domain fand nicht statt.
- Die zehn DFN-Cache-Server wurden untereinander als Parents konfiguriert. Die Requests zwischen den DFN-Cache-Servern wurden in Abhängigkeit der Toplevel Domains gelenkt.

Da die DFN-Cache-Server untereinander als Parents konfiguriert waren, bildeten sie in der gesamten Cache-Hierarchie zwei logische Ebenen, durch die die HTTP-Requests bei Cache-Misses geleitet wurden. Insgesamt ergab sich die in Abbildung 3.2 dargestellte dreistufige Hierarchie im DFN-Cache-Verbund.

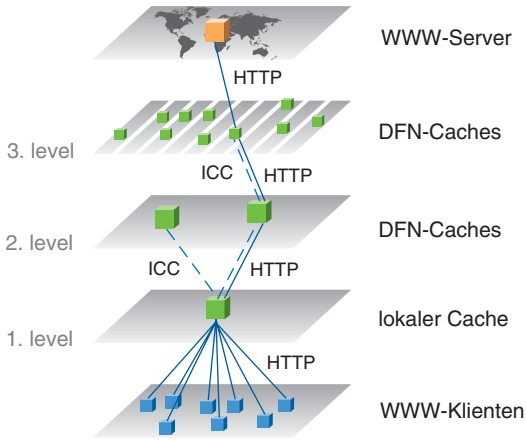


Abbildung 3.2: Dreistufige Cache-Hierarchie – ursprüngliche Konzeption

Bereits nach geringer Betriebsdauer zeigten sich Mängel in der gewählten Konzeption:

- Eine gleichmäßige Verteilung des Verkehrsaufkommens konnte anhand der vorgenommenen Zuweisung von Toplevel Domains nicht erreicht werden. Deutliche Ausprägungen des WWW-Verkehrs aus den Toplevel Domains `com` und `de` zeigten, dass die gewählte breite Verteilung von 50 Toplevel Domains auf die DFN-Cache-Server nicht sinnvoll war (Abbildung 3.3).

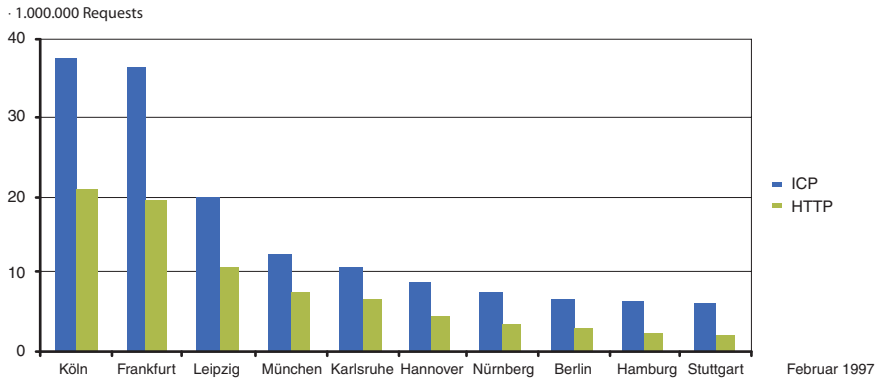


Abbildung 3.3: Verkehrsverteilung auf DFN-Cache-Servern (Februar 1997)

- Das gemessene Aufkommen an WWW-Verkehr aus der Toplevel Domain `com` betrug ca. 50%. Da die Zuordnung von lediglich zwei DFN-Cache-Servern nicht ausreichend war, kam es aufgrund von Überlastung zu häufigen Ausfällen auf den zuständigen DFN-Cache-Servern in Frankfurt und Köln. Besonders deutlich wurde diese Situation, als wegen eines Hardware-Defektes der DFN-Cache-Server in Köln für einen längeren Zeitraum ausfiel. Bereits wenige Minuten nach Beginn des Ausfalls war der DFN-Cache-Server in Frankfurt völlig überlastet, so dass er für die Dauer der Störung in Köln ebenfalls außer Betrieb genommen werden musste.

- Im Fall eines Cache-Miss in der gesamten Cache-Server-Hierarchie wurde ein HTTP-Request unter Umständen durch drei Cache-Server (ein lokaler Cache-Server, zwei DFN-Cache-Server) geleitet. Hierdurch entstand, insbesondere in Verbindung mit überlasteten DFN-Cache-Servern, eine erhebliche Verzögerung, die gegenüber den Nutzern nicht zu vertreten war.

Ausgehend von der dargestellten Situation wurden im Sommer 1997 Überlegungen für eine neue Konzeption des DFN-Cache-Verbundes vorangetrieben.

3.1.2.2 Alter DFN-Cache-Verbund – verbesserte Konzeption

Die Basis für eine neue Konzeption des DFN-Cache-Verbundes bildeten die Analysen der im laufenden Projekt gesammelten Verkehrsstatistiken. Die Statistiken wiesen darauf hin, dass ca. 50% des gesamten WWW-Verkehrs im Wissenschaftsnetz durch Objekte der Toplevel Domain `com` verursacht wurde. Für eine neue Konzeption ergaben sich daraus zwei alternative Ansätze:

Ansatz 1:

- Verzicht auf Partitionierung, d. h. keine domain-abhängige Lenkung der Requests im gesamten Cache-Verbund.
- Alle DFN-Cache-Server werden untereinander als Siblings konfiguriert.
- Jedem lokalen Cache-Server werden zwei DFN-Cache-Server als Parents zugeordnet.

Ansatz 2:

- Der DFN-Cache-Verbund wird in zwei gleiche Hälften partitioniert. Fünf DFN-Cache-Server sind für den Verkehr der Toplevel Domain `com` zuständig, fünf für den restlichen Verkehr der logischen Domain `!com` (nicht `com`).
- Alle DFN-Cache-Server innerhalb einer Partition werden untereinander als Siblings konfiguriert.
- Zwischen den Partitionen werden keine Informationen ausgetauscht.
- Jedem lokalen Cache-Server werden aus jeder Partition zwei DFN-Cache-Server als Parents zugeordnet.

Bei der Entscheidung für eine der genannten Alternativen ist zu beachten, dass Mitte 1997 lediglich ICP als Inter-Cache-Protokoll zur Verfügung stand. Der mit Ansatz 1 dargestellte Verzicht auf eine domain-basierte Lenkung der Requests bedeutet, dass jedes Objekt auf einem beliebigen DFN-Cache-Server gespeichert sein könnte. Daraus folgt, dass ein DFN-Cache-Server bei einem Cache-Miss ICP-Requests an alle weiteren neun DFN-Cache-Server senden muss, um festzustellen, ob das Objekt bereits im DFN-Cache-Verbund gespeichert ist. Dieses hohe Aufkommen an ICP-Verkehr ist nicht zu vertreten.

Mit Ansatz 2 werden statt einer großen Partition aus zehn DFN-Cache-Servern zwei Partitionen mit jeweils fünf DFN-Cache-Servern gebildet. Entsprechend sind bei jedem Cache-Miss ICP-Requests an lediglich vier Neighbors zu versenden. Im Vergleich zu Ansatz 1 wird der auftretende ICP-Verkehr innerhalb des DFN-Cache-Verbundes um mehr als die Hälfte verringert, weshalb die Wahl auf Ansatz 2 fiel. Die Umsetzung dieses Ansatzes erfolgte im Oktober 1997. Die Zuordnung der Domains `com` und `!com` auf die zehn DFN-Cache-Server erfolgte abwechselnd auf dem äußeren Ring des Kernnetzes (Abbildung 3.1 links). In Tabelle 3.4 wird die resultierende Verteilung zusammengefasst.

Toplevel Domain	DFN-Cache-Server
com	Berlin, Frankfurt, Hannover, Nürnberg, Stuttgart
!com	Köln, Hamburg, Karlsruhe, Leipzig, München

Tabelle 3.4: Verteilung der logischen Toplevel Domains `com` und `!com`

Da die DFN-Cache-Server untereinander lediglich als Siblings und nicht als Parents konfiguriert wurden, bildete der DFN-Cache-Verbund mit diesem Ansatz auch bei Cache-Misses lediglich eine Ebene der Cache-Hierarchie. Daraus ergab sich eine Reduzierung der gesamten Cache-Hierarchie von drei auf zwei Stufen (Abbildung 3.4).

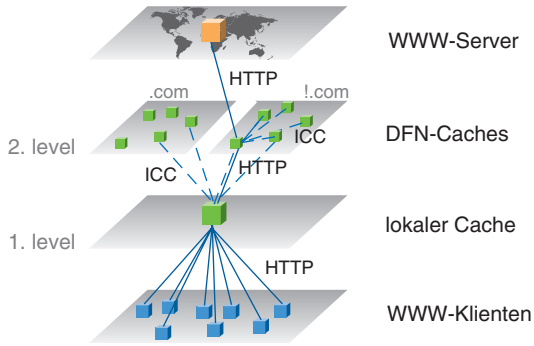


Abbildung 3.4: Zweistufige Cache-Hierarchie – verbesserte Konzeption

Anhand der Verkehrsstatistiken konnten sofort wesentliche Verbesserungen belegt werden (Datensätze 97 und 98 in Tabelle 3.14):

- Das Verkehrsaufkommen zwischen den DFN-Cache-Servern untereinander sank deutlich ab.
- Das von den DFN-Cache-Servern an lokale Cache-Server ausgesendete Datenvolumen konnte von ca. 30 GByte/Tag auf annähernd 60 GByte/Tag verdoppelt werden.
- Die Reduzierung der Cache-Hierarchie auf zwei Stufen führte zu einer Verringerung der durchschnittlichen Übertragungsdauer.

- Der Ausfall eines DFN-Cache-Servers wurden durch die vier verbleibenden DFN-Cache-Server der jeweiligen Partition kompensiert. Somit konnten die im vorangegangenen Kapitel dargelegten Kettenreaktionen nahezu ausgeschlossen werden.

Die Konzeption wurde bis zur Abschaltung des alten DFN-Cache-Verbundes beibehalten. Die Inter-Cache-Kommunikation zwischen den DFN-Cache-Servern wurde im Oktober 1999 von ICP auf Cache Digests umgestellt. Lokale Cache-Server konnten ab Dezember 1999 die Cache Digests der DFN-Cache-Server abrufen.

3.1.2.3 Neuer DFN-Cache-Verbund

Bereits vor der Installation der neuen DFN-Cache-Server im Januar 2000 war ein wesentlicher Teil des Gigabit-Wissenschaftsnetzes (G-WiN) in Betrieb. Aufgrund der zum damaligen Zeitpunkt geringen Auslastung der Strecken im Kernnetz war die Wahl geeigneter Standorte für die neuen DFN-Cache-Server von geringer Bedeutung. Um eine möglichst schnelle Kopplung der DFN-Cache-Server untereinander zu erreichen, wurde die Installation an einem Standort angestrebt. Die Entscheidung fiel auf den Standort Hannover, der den Endpunkt der Transatlantikstrecke bildete (s. Abbildung 3.1).

Mit dem neuen DFN-Cache-Verbund wurde die Konzeption des alten DFN-Cache-Verbundes fortgeführt. Zur Durchführung wurden weniger, jedoch leistungsfähigere DFN-Cache-Server eingesetzt. Da das Volumen des WWW-Verkehrs aus der Toplevel Domain `com` auf mittlerweile unter 50% gesunken war, wurde eine modifizierte Partitionierung gewählt. Mit `abroad` und `!abroad` wurden zwei logische Domains mit jeweils ca. 50% Anteil am Volumen des WWW-Verkehrs eingeführt. Die Domain `abroad` enthielt dabei die Toplevel Domains `com`, `edu`, `gov`, `org`, `int`, `mil`, `us` und `ca`. Die Bezeichnung `abroad` orientierte sich an der Tatsache, dass der WWW-Verkehr aus diesen Toplevel Domains zu einem wesentlichen Teil über die Transatlantikstrecke des G-WiN geführt wurde.

Auf jedem der beiden neuen DFN-Cache-Server konnten parallel zwei voneinander unabhängige Cache-Prozesse ausgeführt werden. Um auch bei Ausfall eines DFN-Cache-Servers beide logischen Toplevel Domains bedienen zu können, wurden jedem DFN-Cache-Server ein Cache-Prozess für die Domain `abroad` und ein Prozess für die Domain `!abroad` zugewiesen. Abbildung 3.5 zeigt die logische Konfiguration der beiden neuen DFN-Cache-Server, wobei die Datenflüsse über die Netzinterfaces `hme0` und `hme1` hervorgehoben sind.

Wie auf den alten DFN-Cache-Servern wurden auch auf den neuen die Prozesse innerhalb einer Partition als Sibling untereinander konfiguriert. Eine Kommunikation zwischen den verschiedenen Partitionen fand nicht statt.

Zur Inter-Cache-Kommunikation wurden im neuen DFN-Cache-Verbund weitgehend Cache Digests verwendet. Einzige Ausnahme bildete die geringe Anzahl loka-

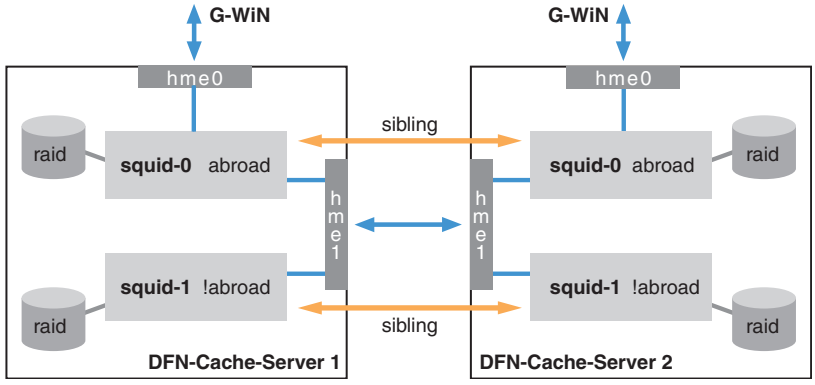


Abbildung 3.5: Konfiguration neue DFN-Cache-Server

ler Cache Server, die nicht auf der Cache-Software Squid basierten und somit auf ICP zurückgreifen mussten.

3.2 Verkehrsflüsse in einem Cache-Verbund

Die Grundlagen für die Berechnung und die Bewertung eines einzelnen Cache-Servers wurden in Kapitel 2.2.2.1 erläutert. Analoge Betrachtungen lassen sich auch auf Cache-Verbünde anwenden, wobei, bedingt durch die Inter-Cache-Kommunikation, zusätzliche Verkehrsflüsse zu berücksichtigen sind [VJBM98] [WVS+99]. In der Literatur sind bisher nur spezielle, auf einzelne Cache-Verbünde angepasste Auswertungen bekannt (z. B. [Wes02] [MVBS98] [JVBM98]). In diesem Kapitel wird ein formal strukturiertes und allgemein anwendbares Verfahren zur Auswertung von Logdateien aus einem Cache-Verbundes entworfen. Es erfolgt eine klare Darlegung der auftretenden Verkehrsflüsse innerhalb des Cache-Verbundes sowie zwischen dem Cache-Verbund und dessen Klienten und den WWW-Servern. Als charakteristische Verkehrsparameter werden sowohl die auftretenden Requests bzw. Responses als auch das übertragene Datenvolumen und die entsprechenden Übertragungszeiten betrachtet. Aus den berechneten Ergebnissen können in einem ersten Schritt Nutzen und Leistungsfähigkeit eines Cache-Verbundes bewertet werden. In einem zweiten Schritt werden die auftretenden Verkehrsflüsse durch analytische Modelle nachgebildet. Diese Modellierung bildet die Grundlage für die in Kapitel 5 vorgestellte Implementierung eines Simulationsmodells von Cache-Verbünden.

3.2.1 Datenquelle

Die für die Analyse der Verkehrsflüsse notwendigen Rohdaten werden auf Cache-Servern in Logdateien protokolliert. Der Aufbau und der Inhalt dieser Logdateien hängt von der jeweiligen Implementierung des Cache-Servers ab, gleichen sich jedoch in den grundlegenden Verkehrsinformationen. Die im Rahmen dieser Arbeit betrachteten Verkehrsstatistiken wurden ausschließlich aus der Logdatei `access.log` der Cache-Software Squid generiert. In dieser Datei werden sämtliche Responses, die ein Cache-Server an seine Klienten sendet, protokolliert. Jede Zeile

der Logdatei steht somit für einen bearbeiteten Request und enthält folgende Informationen [Wes01b]:

- Zeitpunkt, an dem die Verarbeitung eines Requests abgeschlossen wird. Die Angabe des Zeitpunkts erfolgt in Millisekunden seit dem 1. Januar 1970. Zu beachten ist, dass die Angabe des Zeitpunkts je nach Verarbeitung von TCP- oder UDP-Verbindungen unterschiedlich definiert wird:
  - Bei TCP-Verbindungen (z. B. für HTTP) wird der Zeitpunkt protokolliert, an dem der Cache-Server die Verbindung zum Klienten abbaut (Systemaufruf `close()`).
  - Bei UDP-Verbindungen (z. B. für ICP) wird der Zeitpunkt protokolliert, an dem die Response an den Klienten zurückgesendet wird (Systemaufruf `send()`).
- Dauer der Bearbeitung in Millisekunden. Auch hier muss bei der Interpretation zwischen TCP- und UDP-Verbindung unterschieden werden:
  - Die Dauer entspricht bei einer TCP-Verbindung der Zeitspanne zwischen Auf- und Abbau der Verbindung (Systemaufrufe `accept()` and `close()`). Sie enthält somit sowohl die Bearbeitungszeit des Requests auf dem Cache-Server als auch die Übertragungszeiten von Request und Response zwischen Klient und Cache-Server. Da bei Persistent Connections (Kapitel 2.1.3.3) bestehende Verbindungen mehrfach genutzt werden, entspricht hierbei die Dauer lediglich der Bearbeitung des Requests und der Übertragung der Response. Für eine näherungsweise Betrachtung der Übertragungszeiten ist eine undifferenzierte Interpretation jedoch zulässig, da die Übertragung der Requests die gesamte Dauer nur unwesentlich verzögert.
  - Da bei UDP-Verkehr keine Verbindungen etabliert werden, enthält die Dauer lediglich die Verarbeitungszeit des Requests auf dem Cache-Server. Informationen über die Übertragungszeit von Request oder Response können aus diesem Wert nicht abgeleitet werden.
- Umfang der Response in Byte.
- IP-Adresse des Klienten, entspricht der Zieladresse der Response.
- URL.
- Unterscheidung zwischen TCP- und UDP-Verkehr. Die Angabe von UDP-Verkehr ist ein Hinweis auf Inter-Cache-Kommunikation über ICP oder HTCP. TCP-Verkehr kann sowohl die Übertragung von Nutzdaten über z. B. HTTP oder FTP als auch den Austausch von Cache Digests signalisieren. Für eine exakte Klassifizierung ist die Auswertung des URL notwendig.

Bei der Bearbeitung von Nutzdaten werden folgende zusätzlichen Informationen protokolliert:

- Detaillierte Angabe, welche Art von Cache-Hit oder Cache-Miss (s. Kapitel 2.2.2.5) der Request auf dem Cache-Server erzeugt.

- Adresse eines Neighbors oder WWW-Servers. Dieses Feld wird verwendet, wenn der empfangene Request keinen unbestätigten Cache-Hit ergibt und weitergeleitet werden muss.
- Das Kriterium zur Auswahl des geeigneten Neighbors, an den ggf. der Request weitergeleitet wird.
- HTTP-Statuscode der HTTP-Response, die an den Klienten gesendet wird.
- Mimetype des übertragenen Objekts.

Bei der Auswertung der Logdateien ist generell zu beachten, dass nicht jede protokollierte Zeile nur einem Paar von Request und Response entspricht. So bedeutet ein protokollierter Cache-Miss, dass der HTTP-Request eines Klienten empfangen, an einen Neighbor oder WWW-Server weitergeleitet und dessen Response an den Klienten zurückgesendet wurde. Der Eintrag in der Logdatei entspricht somit *zwei* Paaren von Request und Response, die bei der Auswertung des tatsächlich auftretenden Verkehrsaufkommens zu berücksichtigen sind.

3.2.2 Definition der Verkehrsflüsse

Ausgehend von Abbildung 2.7 stellt Abbildung 3.6 die zusätzlichen Verkehrsflüsse, die in der Umgebung eines Cache-Verbundes auftreten, dar.

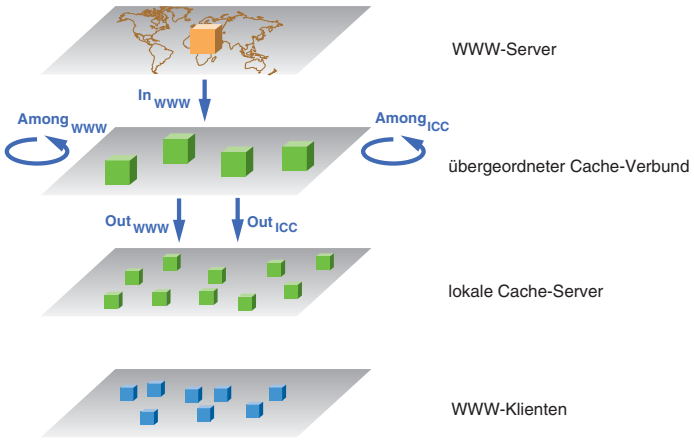


Abbildung 3.6: Verkehrsflüsse in einem Cache-Verbund

Neben der bereits bekannten Übertragung der Nutzdaten In_{WWW} und Out_{WWW} umfasst die Darstellung drei zusätzliche Verkehrsflüsse:

- $Among_{\mathrm{ICC}}$: Inter-Cache-Kommunikation innerhalb des Cache-Verbundes.
- Out_{ICC} : Inter-Cache-Kommunikation zwischen lokalen Cache-Servern und dem Cache-Verbund.
- $Among_{\mathrm{WWW}}$: Übertragung von Nutzdaten innerhalb des Cache-Verbundes.

Welche Bedeutung diesen Verkehrsflüsse beigemessen wird, hängt von den Kosten für die belegten Ressourcen zur Übertragung und den zusätzlichen Latenzen ab, die

durch die Übertragung und Verarbeitung verursacht werden. Aufgrund der damit verbundenen speziellen Randbedingungen sind generelle Angaben für eine vollständige Bewertungen von Cache-Verbünden nicht möglich, sondern nur für Einzelfälle durchführbar. Für die Berechnung der Objekt- und Volumen-Trefferraten von Cache-Verbünden werden in Kapitel 3.3.3 die bereits in Kapitel 2.2.2.1 aufgestellten Gleichungen um die zusätzlichen Verkehrsflüsse erweitert.

In der in Abbildung 3.6 gewählten Darstellung bilden lokale Cache-Server die Klienten des Cache-Verbundes. Es ist durchaus möglich, dass ein Cache-Verbund ausschließlich direkt über WWW-Browser angesprochen wird. Da in diesem Fall keine Inter-Cache-Kommunikation zwischen dem Cache-Verbund und den Klienten auftritt, entfällt hierbei der Verkehrsfluss Out_{ICC}. Dieser Fall wird jedoch in den weiteren Betrachtungen nicht berücksichtigt.

Ausgehend von Abbildung 3.6 lassen sich die in einem Cache-Verbund auftretenden Kommunikationsbeziehungen K formulieren. Die Definitionen der K leiten sich von den unterschiedlichen Quell- und Zieladressen ab, die als Mengen mit den entsprechenden Elementen gemäß Tabelle 3.5 dargestellt werden können.

Menge	Element	Erläuterung
Quelle	local	lokale Cache-Server
	neighbor	Neighbor innerhalb des Cache-Verbundes
Ziel	self	der protokollierende Cache-Server selbst (Hinweis auf unbestätigten Cache-Hit)
	neighbor	Neighbor innerhalb des Cache-Verbundes
	server	originale WWW-Server

Tabelle 3.5: Elemente der Quell- und Zieladressen

Aus den möglichen Kombinationen der Elemente aus den Mengen $Quelle = \{\text{local}, \text{neighbor}\}$ und $Ziel = \{\text{self}, \text{neighbor}, \text{server}\}$ ergeben sich sechs verschiedenen Kommunikationsbeziehungen $K_{q,z}$ mit $q \in Quelle$ und $z \in Ziel$. In Tabelle 3.6 werden die einzelnen Kommunikationsbeziehungen erläutert, wobei der protokollierende Cache-Server rot eingefärbt ist.

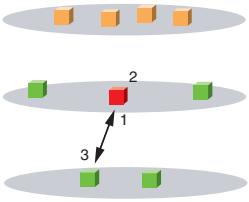
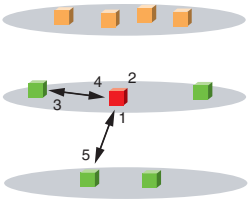
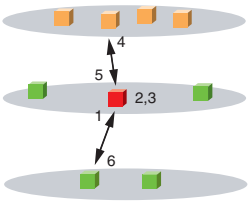
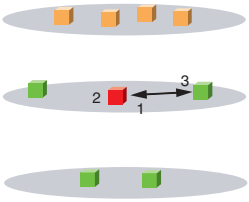
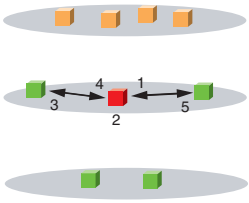
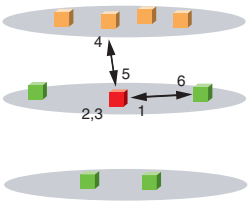
Quelle	Ziel	Darstellung	Kommunikationsablauf
local	*self*		1. Request von lokalem Cache-Server 2. unbestätigter Cache-Hit auf protokollierendem Cache-Server 3. Response ohne weitere Kommunikation mit Neighbors oder originalem WWW-Server
local	*neighbor*		1. Request von lokalem Cache-Server 2. kein unbestätigter Cache-Hit auf protokollierendem Cache-Server 3. Weiterleitung des Requests an Neighbor 4. Response von Neighbor 5. Weiterleitung der Response an lokalen Cache-Server
local	*server*		1. Request von lokalem Cache-Server 2. kein unbestätigter Cache-Hit auf protokollierendem Cache-Server 3. kein Cache-Hit auf Neighbors 4. Weiterleitung des Requests an WWW-Server 5. Response von WWW-Server 6. Weiterleitung der Response an lokalen Cache-Server
neighbor	*self*		1. Request von Neighbor 2. unbestätigter Cache-Hit auf protokollierendem Cache-Server 3. Response ohne Kommunikation mit weiteren Neighbors oder originalem WWW-Server
neighbor	*neighbor*		1. Request von 1. Neighbor 2. kein unbestätigter Cache-Hit auf protokollierendem Cache-Server 3. Weiterleitung des Requests an 2. Neighbor 4. Response von 2. Neighbor 5. Weiterleitung der Response an 1. Neighbor
neighbor	*server*		1. Request von Neighbor 2. kein unbestätigter Cache-Hit auf protokollierendem Cache-Server 3. kein Cache-Hit auf anderen Neighbors 4. Weiterleitung des Requests an WWW-Server 5. Response von WWW-Server 6. Weiterleitung der Response an Neighbor

Tabelle 3.6: Kommunikationsbeziehungen in einem Cache-Verbund

Jede Zeile einer Logdatei kann eindeutig einer der in Tabelle 3.6 aufgeführten Kommunikationsbeziehungen zugeordnet werden. Daraus ergeben sich sechs untereinander disjunkte Mengen, deren Vereinigung nach folgenden Vorschriften die gesuchten Verkehrsflüsse ergibt:

Es gilt:

$$\bigcap K_{q,z} = \varnothing \quad \text{(Kommunikationsbeziehungen sind disjunkt)}$$

$$\bigcup K_{q,z} = \Omega_{\text{Logzeilen}} \quad \text{(Gesamtheit aller Zeilen in Logdateien)}$$

Berechnung:

$$\begin{aligned} In &= \bigcup K_{q,\text{server}} \\ &= K_{\text{local,server}} \cup K_{\text{neighbor,server}} \\ Out &= \bigcup K_{\text{local},z} \\ &= K_{\text{local,self}} \cup K_{\text{local,neighbor}} \cup K_{\text{local,server}} \\ Among &= \bigcup K_{\text{neighbor},z} \\ &= K_{\text{neighbor,self}} \cup K_{\text{neighbor,neighbor}} \cup K_{\text{neighbor,server}} \end{aligned} \tag{3.1}$$

zu beachten:

$$\begin{aligned} In \cap Out &= K_{\text{local,server}} \\ In \cap Among &= K_{\text{neighbor,server}} \\ Out \cap Among &= \varnothing \end{aligned}$$

Zeilen in Logdateien:

$$n_{\text{Logzeilen}} = \sum |K_{q,z}| = |\Omega_{\text{Logzeilen}}|$$

Anzahl Requests:

$$\begin{aligned} n_{\text{Requests}} &= |In| + |Out| + |Among| \\ n_{\text{Requests}} &\geq n_{\text{Logzeilen}} \end{aligned}$$

Die Ungleichung $n_{\text{Requests}} \geq n_{\text{Logzeilen}}$ bestätigt die bereits genannte Feststellung, dass jede Zeile einer Logdatei einem Request (bei unbestätigtem Cache-Hit) oder zwei Requests (in allen anderen Fällen) entsprechen kann.

Die aufgeführten sechs Kommunikationsbeziehungen gelten für die Übertragung von Nutzdaten. Bei der Inter-Cache-Kommunikation werden Informationen direkt von einem Cache-Server abgerufen, so dass als Zieladresse lediglich das Element *self* zu berücksichtigen ist. Die bei der Inter-Cache-Kommunikation zu betrachtenden Kommunikationsbeziehungen können daher auf $K_{\text{local, self}}$ und $K_{\text{neighbor, self}}$ reduziert werden. Weiterhin ist zu beachten, dass $K_{\text{neighbor, neighbor}}$ und $K_{\text{neighbor, server}}$ eine Parent-Beziehung zwischen benachbarten Cache-Servern im Cache-Verbund voraussetzen. Im DFN-Cache-Verbund traten diese Kommunikationsbeziehungen vorrangig in der ursprünglichen Konfiguration (Kapitel 3.1.2.1) auf.

3.2.3 Übertragungszeit und Datenrate

Anhand eines Beispiels wird in Abbildung 3.7 erläutert, wie die in den Logdateien protokollierten Übertragungszeiten in der vorgestellten Klassifizierung berücksichtigt werden.

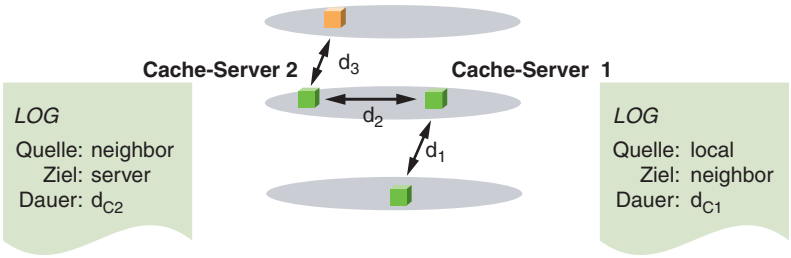


Abbildung 3.7: Bestimmung der Übertragungszeit

In dem Beispiel wird angenommen, dass das geforderte Objekt auf keinem der beteiligten Cache-Server einen unbestätigten Cache-Hit erzeugt und daher vom originalen WWW-Server abgerufen werden muss. Auf beiden Cache-Servern wird dabei je ein Eintrag in der Logdatei protokolliert. Entsprechend den Quell- und Zieladressen wird bei der Auswertung der Eintrag auf dem Cache-Server 1 der Menge $K_{\text{local, neighbor}}$ zugeordnet, der Eintrag auf dem Cache-Server 2 der Menge $K_{\text{neighbor, server}}$. Da aus den Logdateien lediglich zwei Angaben d_{C1} und d_{C2} für die Dauer der Übertragungen zur Verfügung stehen, müssen diese Werte durch sinnvolle Abschätzung den drei tatsächlichen Übertragungszeiten d_1, d_2 und d_3 zugeordnet werden. Hierfür gelten folgende Feststellungen:

- Der Aufbau der Verbindungen erfolgt durch die Klienten, hier zuerst durch den lokalen Cache-Server.
- Der Abbau der Verbindungen erfolgt nach erfolgreicher Übertragung durch die Server, hier zuerst durch den WWW-Server.

Daraus ergibt sich die Ordnung $d_1 > d_2 > d_3$ der einzelnen Übertragungszeiten. Weiterhin entspricht die auf Cache-Server 1 protokollierte Dauer der Übertragungszeit zum lokalen Cache-Server, weshalb $d_{C1} = d_1$ gesetzt werden kann. Eine entsprechende Überlegung für die auf Cache-Server 2 protokollierte Dauer führt zu $d_{C2} = d_2$. Weiterhin gelte die Annahme, dass bei nicht überlasteten und ausreichend dimensionierten Cache-Verbünden die zusätzliche Verzögerung, die durch die interne Kommunikation auftritt, gegenüber den Übertragungszeiten mit den WWW-Servern oder lokalen Cache-Servern vernachlässigt werden kann. Hierdurch ergibt sich die zulässige Näherung $d_2 \approx d_3$.

Aus der im vorangegangenen Abschnitt getroffenen Klassifizierung lässt sich sowohl die Anzahl Requests bzw. Responses als auch das übertragene Datenvolumen direkt berechnen. Die Ergebnisse liefern die notwendigen Daten zur Feststellung der Einsparungen und Leistungsfähigkeit eines Cache-Verbundes. Ein entsprechendes Vorgehen bei der Interpretation der Übertragungszeiten ist jedoch nicht möglich, da

die Dauer wesentlich von dem übertragenen Datenvolumen abhängt. Die korrekte Darstellung bei erfolgreicher Übertragung von Objekten führt zu der Angabe von mittleren Datenraten, die sich als Quotienten aus Datenvolumen und Übertragungszeit ergeben.

In anderen Fällen wird die Dauer einer Übertragung durch systembedingte Fehler auf den Servern verzögert. So kann ein Server den Versuch eines Verbindungsaufbaus oder eines Dateizugriffs nach fest vorgegebenen Zeitintervallen abbrechen und dem Klienten den aufgetretenen Fehler mit einem Statuscode 5xx in der HTTP-Response signalisieren. Da die Zeitintervalle mehrere Minuten betragen können, ergeben sich hierbei zwar geringe mittlere Datenraten, jedoch erhebliche konstante Belegungsdauern, in denen Ressourcen auf den Servern beansprucht werden. Die in Kapitel 4.1 durchgeführte Analyse und Modellierung der Übertragungszeiten erfordert daher eine differenzierte Betrachtung zwischen der erfolgreichen Übertragung von Nutzdaten und dem Verhalten bei Fehlern.

Bei der Übertragung von Nutzdaten müssen in einem weiteren Schritt die verschiedenen Arten von Cache-Hits und Cache-Misses nach Tabelle 2.1 berücksichtigt werden. Dabei ist der Umgang sowohl mit veralteten Objekten auf den Cache-Servern als auch mit bedingten HTTP-Requests der Klienten (`If-Modified-Since:`, s. Kapitel 2.2.2.5) zu beachten, da hierdurch unterschiedliche Datenvolumen und Übertragungsraten auf den jeweiligen Übertragungsstrecken auftreten. Die Auswirkung dieser Unterschiede wird bei der Modellierung in Kapitel 4 deutlich, im Folgenden wird zunächst das Vorgehen bei der Analyse erläutert.

In Tabelle 3.7 sind die verschiedenen Möglichkeiten aufgeführt, nach denen ein HTTP-Request mit der Methode `GET` zwischen Quelle und Ziel über einen Cache-Server abgearbeitet werden kann.

Quelle	{local, neighbor}			
HTTP-Request von Quelle	GET			
Objekt auf Cache-Server	vorhanden und aktuell	vorhanden, aber veraltet		nicht vorhanden
Ziel	{self}	{neighbor, server}		
HTTP-Request an Ziel		IMS		GET
HTTP-Response von Ziel		200	304	200
HTTP-Response an Quelle	200	200	200	200

Tabelle 3.7: Varianten bei der Bearbeitung eines HTTP-Requests

Unter Berücksichtigung der verschiedenen Elemente von Quelle und Ziel ergeben sich insgesamt 14 Kombinationen, die theoretisch bei der Interpretation der Datenrate unterschieden werden müssen. Zu einer noch höheren Zahl von Kombinationen führt die in Tabelle 3.8 dargestellte Bearbeitung von bedingten HTTP-Requests, da

in den HTTP-Responses statt des Objekts auch lediglich der Statuscode 304 (`not-modified`) übertragen werden kann.

Quelle	{local, neighbor}							
HTTP-Request von Quelle	IMS							
Objekt auf Cache-Server	vorhanden und aktuell		vorhanden, aber veraltet				nicht vorhanden	
Ziel	{self}		{neighbor, server}					
HTTP-Request an Ziel			IMS				IMS	
HTTP-Response von Ziel			200		304		200	304
Objekt auf Quelle	alt	aktuell	alt	aktuell	alt	aktuell	alt	aktuell
HTTP-Response an Quelle	200	304	200	304	200	304	200	304

Tabelle 3.8: Varianten bei der Bearbeitung eines bedingten HTTP-Requests

Es ergeben sich 28 weitere Kombinationen, so dass bei der Übertragung von Nutzdaten in einem Cache-Verbund theoretisch insgesamt 42 verschiedene Möglichkeiten von HTTP-Verbindungen über einen einzelnen Cache-Server zu betrachten sind. Durch Zusammenfassen von Kombinationen kann jedoch eine Vereinfachung durchgeführt werden. So wird die Datenrate bei der Übertragung von Nutzdaten wesentlich durch die HTTP-Response bestimmt, weshalb die Art des HTTP-Request vernachlässigt werden kann. Zudem unterscheidet sich ein bedingter HTTP-Request von einem normalen HTTP-Request nur durch die Zeile `If-Modified-Since:`. Es bietet sich daher an, eine Differenzierung nur anhand der Statuscodes in den HTTP-Responses durchzuführen. Die Anzahl der Kombinationen kann dadurch auf die in Tabelle 3.9 dargestellten 20 Möglichkeiten reduziert werden.

Quelle	{local, neighbor}					
Ziel	{self}		{neighbor, server}			
HTTP-Response von Ziel			304		200	
HTTP-Response an Quelle	304	200	304	200	304	200

Tabelle 3.9: Zusammenfassung verschiedener HTTP-Codes

3.2.4 Cache-Hit und Cache-Miss

In Tabelle 3.6 wurden die elementaren Kommunikationsbeziehungen aufgeführt, nach denen die in einer Logdatei protokollierten Einträge klassifiziert werden können. Durch die Kombination verschiedener Kommunikationsbeziehungen ergibt sich eine differenzierte Betrachtung von Cache-Hits und Cache-Misses. Die in einem Cache-Verbund auftretenden elementaren vier Kombinationen werden in

Tabelle 3.10 dargestellt, die protokollierenden Cache-Server C_1 und C_2 sind rot eingefärbt.

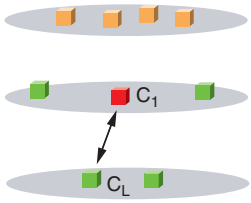
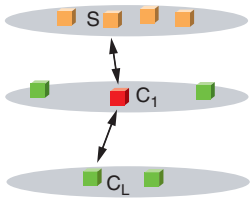
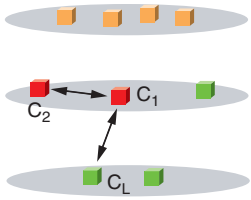
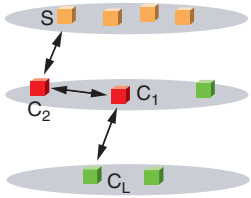
Cache-Hit / Cache-Miss	Darstellung	protokollierte Kommunikation-beziehung	Statuscodes Nutzdaten			
			C_L–C_1	C_1–S	C_1–C_2	C_2–S
Hit		C_1: $K_{\text{local, neighbor}}$	200	-	-	-
			304	-	-	-
Miss		C_1: $K_{\text{local, server}}$	200	200	-	-
			200	304	-	-
			304	200	-	-
			304	304	-	-
Neighbor-Hit		C_1: $K_{\text{local, neighbor}}$ C_2: $K_{\text{neighbor, self}}$	200	-	200	-
			200	-	304	-
			304	-	200	-
			304	-	304	-
Neighbor-Miss		C_1: $K_{\text{local, neighbor}}$ C_2: $K_{\text{neighbor, server}}$	200	-	200	200
			200	-	200	304
			200	-	304	200
			200	-	304	304
			304	-	200	200
			304	-	200	304
			304	-	304	200
			304	-	304	304

Tabelle 3.10: Cache-Hits und Cache-Misses in einem Cache-Verbund

Zu beachten ist, dass ein Neighbor-Miss nur in zwei Situationen auftreten kann:

1. bei Parent-Beziehungen zwischen den Cache-Servern innerhalb des Cache-Verbundes oder

2. bei falschen Cache-Hits auf dem Cache-Server C_2 (Kapitel 2.3.1).

Nach Kapitel 3.2.3 ist bei der Betrachtung von Nutzdaten zu unterscheiden, ob tatsächlich das Objekt übertragen wird (Statuscode 200) oder lediglich eine Bestätigung der Aktualität erfolgt (Statuscode 304). Entsprechend gibt es für jede der in Tabelle 3.10 dargestellten HTTP-Responses jeweils zwei Möglichkeiten von Statuscodes, woraus sich die im rechten Teil der Tabelle aufgeführten Kombinationen ergeben. Die zehn Kombinationen für Hit, Miss und Neighbor-Hit entsprechen den bereits in Tabelle 3.9 dargestellten zehn Kombinationen bei der Kommunikation zwischen lokalen und übergeordneten Cache-Servern.

Im Gegensatz hierzu ist für die Übertragung von Kontrollnachrichten keine differenzierte Betrachtung der verschiedenen HTTP-Responses erforderlich. Entsprechend ergeben sich für die HTTP-Responses mit diesen Statuscodes lediglich die vier in Tabelle 3.10 dargestellten Kombinationen von Cache-Hits oder Cache-Misses.

3.2.5 Berechnung der Inter-Cache-Kommunikation

Das Verkehrsaufkommen zur Inter-Cache-Kommunikation in einem Cache-Verbund lässt sich sowohl für ICP als auch für Cache Digests in Abhängigkeit der übertragenen Nutzdaten berechnen. Neben den spezifischen Eigenschaften der verwendeten Protokolle hängt die Anzahl der auftretenden Requests wesentlich von der Struktur des übergeordneten Cache-Verbundes und der Anzahl der beteiligten Cache-Server ab. Für die Formulierung der Gleichungen werden zunächst folgende Parameter definiert, wobei ein nach Abbildung 3.6 partitionierter Cache-Verbund vorausgesetzt wird:

- L : Anzahl der lokalen Cache-Server.
- P: Anzahl der Partitionen im übergeordneten Cache-Verbund.
- c_j mit $j = \{1,\ldots,P\}$: Anzahl der übergeordneten Cache-Server in Partition j .
- $n_{i,j}$ mit $i = \{1,\ldots,L\}$ und $j = \{1,\ldots,P\}$: Anzahl der übergeordneten Cache-Server (Neighbors) aus der Partition j, die von dem lokalen Cache-Server i befragt werden.

Wird vorausgesetzt, dass jeder übergeordnete Cache-Server genau einer Partition zugeordnet ist, dann gilt für die Anzahl C der übergeordneten Cache-Server:

$$C = \sum_{j=1}^{P} c_j \tag{3.2}$$

Die Herleitungen zur Berechnung der Inter-Cache-Kommunikation werden für ICP und Cache Digests getrennt durchgeführt. Dabei werden die Gleichungen lediglich für die auftretenden Request formuliert. In Verbindung mit den entsprechenden Verteilungsfunktionen der Objektgrößen ergibt sich daraus das mit den jeweiligen Pro-

tokollen übertragene Datenvolumen. Es ist zu beachten, dass die Angaben für die Berechnung von ICP-Verkehr ohne Änderungen auch für HTCP-Verkehr gelten.

3.2.5.1 ICP

Nach Kapitel 2.3.3.1 tritt bei einer Parent-Beziehung zwischen zwei Cache-Servern ICP-Verkehr stets in Verbindung mit einem nachfolgenden HTTP-Request auf. Daraus ergeben sich für die Verkehrsflüsse in einem Cache-Verbund (Abbildung 3.6) die Abhängigkeiten $R_{Out_{\mathrm{ICP}}} \sim R_{Out_{\mathrm{WWW}}}$ und $R_{Among_{\mathrm{ICP}}} \sim R_{Among_{\mathrm{WWW}}}$.

Zur Herleitung einer vollständigen Gleichung für $R_{Out_{\mathrm{ICP}}}$ wird zunächst ein einzelner HTTP-Request eines lokalen Cache-Servers l betrachtet. Dieser HTTP-Request soll an einen Neighbor aus der Partition p des übergeordneten Cache-Verbundes geleitet werden. Die Auswahl der geeigneten Partition p erfolgt z. B. nach domain-basierter Auswertung der URL. Für die Bestimmung des geeigneten Neighbors aus p werden vor der Übertragung des HTTP-Requests zunächst ICP-Requests an alle $n_{l,p}$ Neighbors von l in p gesendet. Daraus folgt, dass jedem der $R_{Out_{\mathrm{WWW}}}(l,p)$ HTTP-Requests von l an Neighbors in p genau $R_{Out_{\mathrm{WWW}}}(l,p) \cdot n_{l,p}$ ICP-Requests vorausgehen. Die Summe über alle Partitionen P ergibt somit die Anzahl $R_{Out_{\mathrm{ICP}}}(l)$ aller ICP-Requests zwischen dem lokalen Cache-Server l und dem Cache-Verbund:

$$R_{Out_{\mathrm{ICP}}}(l) = \sum_{j=1}^{P} R_{Out_{\mathrm{WWW}}}(l,j) \cdot n_{l,j} \tag{3.3}$$

Die gesamte Anzahl aller ICP-Requests $R_{Out_{\mathrm{ICP}}}$ ergibt sich direkt aus Gleichung 3.3 durch die Summe über alle lokalen Cache-Server L:

$$R_{Out_{\mathrm{ICP}}} = \sum_{i=1}^{L} \sum_{j=1}^{P} R_{Out_{\mathrm{WWW}}}(i,j) \cdot n_{i,j} \tag{3.4}$$

Die Schwierigkeit bei der Berechnung nach Gleichung 3.4 liegt in der aufwändigen Bestimmung von $R_{Out_{\mathrm{WWW}}}(i,j)$. Eine Vereinfachung von Gleichung 3.4 ergibt sich dann, wenn jedem lokalen Cache-Server die gleiche Anzahl Neighbors n aus jeder Partition zugewiesen wird. Unter dieser Bedingung kann Gleichung 3.4 auf eine einfache Beziehung reduziert werden:

$$\begin{aligned} &\text{für} \quad n_{i,j} = n \quad \forall i = \{1,\ldots,L\} \quad \text{und} \quad j = \{1,\ldots,P\} \\ &R_{Out_{\mathrm{ICP}}} = n \cdot \sum_{i=1}^{L} \sum_{j=1}^{P} R_{Out_{\mathrm{WWW}}}(i,j) = n \cdot R_{Out_{\mathrm{WWW}}} \end{aligned} \tag{3.5}$$

Die bisher in diesem Kapitel formulierten Gleichungen gelten ausschließlich für Parent-Beziehungen zwischen Cache-Servern, da hierbei ein wechselseitiger Zusammenhang zwischen ICP- und HTTP-Requests besteht. Für Sibling-Beziehungen gelten diese Gleichungen jedoch nicht. Bei Sibling-Beziehungen gehen zwar ebenfalls

jedem HTTP-Request $n_{l,p}$ ICP-Requests an die Neighbors voraus, umgekehrt folgt auf diese ICP-Requests jedoch nur dann ein HTTP-Request an einen Neighbor, wenn mindestens ein ICP-Hit erzeugt wurde. Mit der oben formulierten Vereinfachung ergibt sich daher für Sibling-Beziehungen aus Gleichung 3.5 lediglich die Relation $R_{Out_{\mathrm{ICP}}} \gg n \cdot R_{Out_{\mathrm{WWW}}}$. Eine von den übertragenen Nutzdaten $R_{Out_{\mathrm{WWW}}}(i,j)$ ausgehende exakte Berechnung von $R_{Out_{\mathrm{ICP}}}$ ist für Sibling-Beziehungen nicht möglich.

Gleichung 3.4 kann unter Verwendung von $R_{Among_{\mathrm{WWW}}}(i,j)$ auch für die Berechnung von $R_{Among_{\mathrm{ICP}}}(i,j)$ angewendet werden, wenn ausschließlich Parent-Beziehungen zwischen den übergeordneten Cache-Servern bestehen. Durch modifizierte Definitionen von L und P lassen sich sogar beliebige Teilmengen von $R_{Among_{\mathrm{ICP}}}(i,j)$ berechnen. Der ICP-Verkehr innerhalb einer Partition p mit L_p Neighbors ergibt sich zum Beispiel aus:

$$R_{Among_{\mathrm{ICP}}}(p) = \sum_{i=1}^{L_p} R_{Among_{\mathrm{WWW}}}(i,p) \cdot n_{i,p} \tag{3.6}$$

Als weiteres Beispiel wird in Kapitel 5.3.4 die Berechnung des ICP-Verkehrs innerhalb eines Cache-Verbundes in Abhängigkeit der Requests-Trefferraten durchgeführt.

3.2.5.2 Cache Digests

Cache Digests werden nach Kapitel 2.3.3.3 periodisch nach fest vorgegebenen Zeitintervallen abgerufen. Die Anzahl der übertragenen Cache Digests hängt daher nicht von den übertragenen Nutzdaten ab, sondern wird lediglich durch die Parameter L, P und $n_{i,j}$ sowie die Häufigkeit bestimmt, mit der die beteiligten Cache-Server ihre Cache Digests untereinander austauschen. Diese Häufigkeit kann als eine für alle beteiligten Cache-Server konstante Frequenz f betrachtet werden. In der Cache-Software Squid beträgt der für f vorgegebene Wert 1 Cache Digest pro Stunde. Mit dieser Frequenz werden die Cache Digests von jedem Neighbor abgerufen. Durch die Frequenz fließt ein Zeitbezug in die Berechnungen ein, so dass die Anzahl der auftretenden Cache Digests nur in Abhängigkeit eines betrachteten Zeitintervalls t angegeben werden kann.

Analog zu den Überlegungen bei der Berechnung des ICP-Verkehrs ergibt sich die Anzahl der Cache Digests, die ein lokaler Cache-Server l während der Zeit t abruft, aus der Summe der $n_{l,j}$ Neighbors über alle Partitionen P:

$$R_{Out_{\mathrm{Cache\ Digests}}}(l,t) = t \cdot f \cdot \sum_{j=1}^{P} n_{l,j} \tag{3.7}$$

Die Anzahl $R_{Out_{\mathrm{Cache\ Digests}}}$ der insgesamt abgerufenen Cache Digests ergibt sich entsprechend aus der Summe über alle lokalen Cache-Server L:

$$R_{Out_{\text{Cache Digests}}}(t) = t \cdot f \cdot \sum_{i=1}^{L} \sum_{j=1}^{P} n_{i,j} \tag{3.8}$$

Auch hier ergibt sich eine Vereinfachung, wenn den lokalen Cache-Servern aus jeder Partition die gleiche Anzahl Neighbors n zugewiesen wird. Dann reduziert sich Gleichung 3.8 zu:

$$\begin{gathered} \text{für } n_{i,j} = n \quad \forall i = \{1,\ldots,L\} \text{ und } j = \{1,\ldots,P\} \\ R_{Out_{\text{Cache Digests}}}(t) = t \cdot f \cdot L \cdot P \cdot n \end{gathered} \tag{3.9}$$

Gleichung 3.8 kann ohne Veränderung auch zur Berechnung der innerhalb eines Cache-Verbundes ausgetauschten Cache Digests $R_{Among_{\text{Cache Digests}}}(t)$ angewandt werden. Eine Vereinfachung ergibt sich hier, wenn innerhalb jeder Partition alle Cache-Server ihre Cache Digests untereinander austauschen und kein Austausch zwischen den Partitionen stattfindet. Daraus folgt für eine Partition p mit n_p Cache-Servern, dass in jedem Zeitintervall $t = 1/f$ genau $n_p \cdot (n_p - 1)$ Cache Digests übertragen werden. Das gesamte Aufkommen an Cache Digests innerhalb des Cache-Verbundes berechnet sich dann aus der Summe über alle Partitionen P:

$$R_{Among_{\text{Cache Digests}}}(t) = t \cdot f \cdot \sum_{j=1}^{P} n_j \cdot (n_j - 1) \tag{3.10}$$

3.3 Verkehrsanalysen im DFN-Cache-Verbund

Die Verkehrsflüsse im DFN-Cache-Verbund wurden über den gesamten Projektzeitraum kontinuierlich überwacht und protokolliert. Neben dem Monitoring, welches durch ständige Abfragen über das Simple Network Management Protocol (SNMP) und den von der Cache-Software Squid zur Verfügung gestellten Cache Manager aktuelle Zustandsinformationen der DFN-Cache-Server lieferte, wurden täglich die gesamten Logdateien aller DFN-Cache-Server zentral am RVS gesammelt und ausgewertet. Auf Basis der Logdateien wird in diesem Kapitel eine Verkehrsanalyse des DFN-Cache-Verbundes durchgeführt.

3.3.1 Betrachtete Zeiträume

Eine Interpretation der Verkehrsanalyse kann nicht ohne Berücksichtigung der jeweils geltenden Rahmenbedingungen erfolgen. Da temporäre Ausfälle oder Umkonfigurationen die Verkehrsströme innerhalb eines Cache-Verbundes maßgeblich beeinflussen, wurden aus dem gesamten Projektzeitraum fünf repräsentative Ausschnitte zu je 28 Tagen untersucht, in denen ein stabiler Betrieb des DFN-Cache-Verbundes gegeben war. In Tabelle 3.11 sind die gewählten Zeiträume der Datensätze mit den jeweils verwendeten Versionen von Squid, den auf den Cache-Servern

installierten Betriebssystemen sowie die Anzahl der den DFN-Cache-Verbund nutzenden lokalen Cache-Server aufgeführt.

ID	Zeitraum	Version von Squid	Betriebssystem	Klienten
97	1.2. – 28.2.1997	1.1.5	Solaris 2.5.1	40
98	1.2. – 28.2.1998	1.1NOVM20	Solaris 2.5.1	151
99	1.2. – 28.2.1999	2.1patch2	Solaris 2.5.1	180
00	1.2. – 28.2.2000	2.3stable1	Solaris 7	165
01	14.11. – 12.12.2000	2.3stable4	Solaris 7	73

Tabelle 3.11: Aufstellung der betrachteten Zeiträume

Bei der Angabe der Klienten wurden nur die lokalen Cache-Server berücksichtigt, die an mindestens 27 Tagen mindestens 500 HTTP-Requests pro Tag an den DFN-Cache-Verbund stellten.

Die Auswahl der Zeiträume wurde auch mit dem Ziel vorgenommen, möglichst unterschiedliche Kombinationen von Hardware, Konzeption und eingesetzten Protokollen zur Inter-Cache-Konfiguration darzustellen (s. Kapitel 3.1). Aus Tabelle 3.12 ist zu entnehmen, dass lediglich die Datensätze 98 und 99 unter gleichen Rahmenbedingungen protokolliert wurden.

ID	Hardware der DFN-Cache-Server	Konzeption	Inter-Cache-Kommunikation	
			innerhalb DFN-Cache-Verbund	mit lokalen Cache-Servern
97	alt	alt	ICP	ICP
98	alt	neu	ICP	ICP
99	alt	neu	ICP	ICP
00	alt	neu	Cache Digests	ICP / Cache Digests
01	neu	–	Cache Digests	Cache Digests

Tabelle 3.12: Konfiguration des DFN-Cache-Verbundes

In Tabelle 3.13 sind sämtliche Zeilen und Verkehrsflüsse summiert, die aus den Logdateien der jeweiligen Zeiträume fehlerfrei extrahiert werden konnten. Diese Zahlen bilden die Grundlage aller im Rahmen dieser Arbeit durchgeführten Verkehrsanalysen.

ID	Zeilen	Responses	Volumen [Byte]	Übertragungsdauer [ms]
97	200.855.485	226.904.778	1.174.464.834.271	842.282.438.437
98	771.638.528	844.081.642	2.471.332.322.199	1.268.458.899.080
99	1.152.311.233	1.261.309.166	3.387.239.066.961	2.403.944.562.681
00	798.142.282	913.894.996	3.920.784.956.841	1.541.133.669.570
01	346.773.607	537.177.943	8.272.515.983.003	1.135.730.642.067

Tabelle 3.13: In Beobachtungszeiträumen insgesamt protokollierte Replies

Die Auswertungen in den folgenden Kapiteln liefern eine detaillierte Aufschlüsselung der einzelnen Verkehrsflüsse. Bei der Interpretation ist zu beachten, dass der Verkehrsfluss Out_{WWW} die Nutzung des DFN-Cache-Verbundes durch die lokalen Cache-Server wiedergibt. Die Verkehrsflüsse $Among$ und Out_{ICC} werden wesentlich durch die Konfiguration des DFN-Cache-Verbundes (Tabelle 3.12) bestimmt. Das Verkehrsaufkommen von In_{WWW} resultiert aus den im DFN-Cache-Verbund nicht erfolgreich bearbeiteten Requests und der Leistungsfähigkeit der DFN-Cache-Server. Die Übertragungsdauer von In_{WWW} ergibt sich jedoch auch aus nicht beeinflussbaren Faktoren, wie den Leitungskapazitäten auf den externen Anbindungen des Wissenschaftsnetzes.

3.3.2 Verkehrsflüsse

3.3.2.1 WWW-Verkehr

In Tabelle 3.14 werden die übertragenen Nutzdaten aufgeführt. Zu erkennen ist ein kontinuierlicher Anstieg der bearbeiteten Requests und des übertragenen Datenvolumens. Demgegenüber sinkt die gesamte Übertragungsdauer in den letzten beiden betrachteten Zeiträumen deutlich ab. Hier ist ein erster Hinweis darauf zu sehen, das durch geeignete Maßnahmen zur Konfiguration und besonders durch die leistungsfähigere Hardware im neuen DFN-Cache-Verbund eine höhere Dienstqualität erzielt werden konnte. Aus der ebenfalls reduzierten Dauer von In_{WWW} ist weiterhin abzulesen, dass die externen Anbindungen des Wissenschaftsnetzes und allgemein die Erreichbarkeit der WWW-Server im Internet verbessert wurde.

	ID	Requests	Volumen [Byte]	Dauer [ms]
Out_{WWW}	97	32.760.149	489.588.640.727	366.795.922.684
	98	123.250.884	1.380.262.169.346	664.382.817.382
	99	181.147.786	1.877.555.322.385	1.266.817.588.116
	00	170.790.011	2.056.920.809.179	821.955.865.449
	01	287.872.746	4.536.482.938.027	590.026.917.842
$Among_{WWW}$	97	17.620.065	278.287.510.489	182.817.997.378
	98	11.027.517	127.476.036.030	17.346.906.047
	99	16.753.406	166.429.987.936	39.985.904.301
	00	21.914.695	138.512.330.076	33.819.399.324
	01	1.743.440	30.494.432.253	2.596.917.466
In_{WWW}	97	25.581.153	396.315.845.303	292.163.413.147
	98	74.300.274	918.845.988.158	585.722.092.254
	99	109.602.374	1.274.679.192.720	1.097.136.741.526
	00	116.589.181	1.664.175.953.076	683.239.038.215
	01	190.637.648	3.158.676.032.955	540.191.020.449

Tabelle 3.14: Verkehrsflüsse Nutzdaten

3.3.2.2 ICP

Die Auswertung der Inter-Cache-Kommunikation erfordert eine differenzierte Betrachtung von ICP und Cache Digests. Aus dem in Tabelle 3.15 aufgeführten ICP-Verkehr ist abzulesen, dass durch die Verwendung von Cache Digests der interne ICP-Verkehr $Among_{\mathrm{ICP}}$ bereits Anfang 2000 (Datensatz 00), der externe ICP-Verkehr Out_{ICP} Ende 2000 (Datensatz 01) deutlich reduziert werden konnte. Zu beachten ist auch, dass mit der Umstellung auf die neue Konzeption im alten DFN-Cache-Verbund das Aufkommen an internem ICP-Verkehr zwischen 1997 und 1998 wesentlich anstieg.

	ID	Requests	Volumen [Byte]	Dauer [ms]
Out_{ICP}	97	118.947.073	8.078.264.255	401.249.613
	98	420.106.177	29.468.226.691	664.540.245
	99	607.318.876	43.587.771.530	2.663.405
	00	604.079.136	45.831.320.661	263.268.570
	01	56.410.993	4.270.688.477	24.714.985
$Among_{ICP}$	97	31.996.338	2.194.573.497	103.855.615
	98	215.396.790	15.279.901.974	342.543.152
	99	346.486.724	24.986.792.390	1.665.333
	00	428.403	31.534.742	584.005
	01	339.245	21.900.138	0

Tabelle 3.15: Verkehrsflüsse ICP

Nach Kapitel 3.2.5 kann eine mathematische Beziehung zwischen den Requests von Out_{WWW} und Out_{ICP} aufgestellt werden. Entsprechend der überarbeiteten Konzeption im alten DFN-Cache-Verbund sollte in den Zeiträumen 98, 99 und 00 jeder lokale Cache-Server je zwei DFN-Cache-Server aus beiden Partitionen als Parent nutzen. Daher kann für die Berechnung von $R_{out_{\mathrm{ICP}}}$ Gleichung 3.5 mit $n = 2$ angewendet werden. Die Gegenüberstellung der berechneten Ergebnisse mit den gemessenen Werten wird in Tabelle 3.16 dargestellt.

ID	$R_{out_{\mathrm{WWW}}}$	$R_{out_{\mathrm{ICP}}}$		Abweichung
		berechnet	gemessen	
98	123.250.884	246.501.768	420.106.177	+70%
99	181.147.786	362.295.572	607.318.876	+68%
00	170.790.011	341.580.022	604.079.136	+77%

Tabelle 3.16: Vergleich der berechneten und gemessenen ICP-Requests

Aus der hohen Abweichung von 68–77% ist zu erkennen, dass entgegen der vereinbarten Konzeption zahlreiche lokale Cache-Server die DFN-Cache-Server lediglich als Siblings und nicht als Parents nutzten. Da diese lokalen Cache-Server ausschließlich Cache-Hits auf den DFN-Cache-Servern erzeugten, konnten sie einfach identifiziert und aus den Berechnungen eliminiert werden. Die auf die Betrachtung

von Parent-Beziehungen korrigierten Ergebnisse zeigt Tabelle 3.17. Mit den aufgeführten Ergebnissen werden die Gleichungen aus Kapitel 3.2.5.1 bestätigt.

ID	$R_{out_{\text{WWW}}}$	$R_{out_{\text{ICP}}}$		Abweichung
		berechnet	gemessen	
98	75.143.523	150.287.046	157.064.992	+4,51%
99	132.649.021	265.298.042	272.142.731	+2,58%
00	118.124.523	236.249.046	245.132.010	+3,76%

Tabelle 3.17: Korrigierter Vergleich ICP-Requests

Für die Datensätze 97 und 01 kann Gleichung 3.5 zur Berechnung von $R_{out_{\text{ICP}}}$ nicht angewendet werden. Für den Zeitraum 97 mit der ursprünglichen Konzeption im alten DFN-Cache-Verbund muss eine partitions-abhängige Berechnung der ICP-Requests anhand Gleichung 3.4 erfolgen, da $n_{i,j} \neq n$ gilt. Die für die Berechnung notwendigen Werte der HTTP-Requests $R_{Out_{\text{WWW}}}(i,j)$ liegen jedoch nicht vor. Im Zeitraum 01 wurden primär Cache Digests zur Inter-Cache-Kommunikation eingesetzt, ICP-Verkehr trat nur in Ausnahmefällen auf. Eine Berechnung von $R_{Among_{\text{ICP}}}$ kann für keinen der betrachteten Zeiträume erfolgen, da stets Sibling-Beziehungen zwischen den DFN-Cache-Servern verwendet wurden.

3.3.2.3 Cache Digests

Der Einsatz von Cache Digests beschränkte sich auf die beiden letzten untersuchten Zeiträume 00 und 01. Aus Tabelle 3.18 kann zunächst ein durchschnittliches Volumen der Cache Digests von 160 KByte für den Zeitraum 00 und von 3 MByte für den Zeitraum 01 abgelesen werden. Der deutliche Unterschied lässt sich mit der wesentlich höheren Anzahl an Objekten, die auf den neuen DFN-Cache-Servern gespeichert wurden, erklären.

	ID	Requests	Volumen [Byte]	Dauer [ms]
$Out_{CacheDigests}$	00	65.788	10.700.999.330	1.177.981.164
	01	171.153	533.994.774.274	2.880.003.173
$Among_{CacheDigests}$	00	27.782	4.612.009.777	677.532.843
	01	2.718	8.575.216.879	11.068.152

Tabelle 3.18: Verkehrsflüsse Cache Digests

Die Werte der in Tabelle 3.18 aufgeführten Requests lassen sich mit den Gleichungen aus Kapitel 3.2.5.2 überprüfen, wobei eine Dauer der betrachteten Zeiträume von t = 28 Tagen = 672 Stunden gilt. Aus den Erläuterungen in Kapitel 3.1.2.2 und

Kapitel 3.1.2.3 folgt, dass der DFN-Cache-Verbund in beiden Zeiträumen aus $P = 2$ Partitionen bestand und lokale Cache-Server die Cache Digests von jeweils zwei DFN-Cache-Servern aus jeder Partition abriefen. Mit $n_1 = n_2 = 2$ kann zur Berechnung von $R_{Out_{\text{Cache Digests}}}$ die vereinfachte Gleichung 3.9 angewendet werden. Dabei ist jedoch zu beachten, dass nicht alle der in Tabelle 3.11 aufgeführten Klienten Cache Digests zur Inter-Cache-Kommunikation nutzen konnten. Im Zeitraum 00 wurde der Einsatz von Cache Digests zwischen lokalen Cache-Servern und dem DFN-Cache-Verbund experimentell durchgeführt. Aus einer getrennten Auswertung folgte, dass lediglich $L_{00} = 23$ Cache-Server über den gesamten Zeitraum Cache Digests von den DFN-Cache-Servern abriefen. Demgegenüber wurden im Zeitraum 01 Cache Digests bevorzugt eingesetzt. Auf neun lokalen Cache-Servern konnte jedoch nicht die Cache-Software Squid eingesetzt werden, die Voraussetzung für die Übertragung von Cache Digests ist. Dadurch reduzierte sich die Anzahl der Klienten zur Berechnung von $R_{Out_{\text{Cache Digests}}}$ auf $L_{01} = 64$. Aus diesen Angaben konnte die Berechnung vollständig durchgeführt werden, in Tabelle 3.19 sind die gemessenen und berechneten Werte gegenübergestellt.

ID	gemessen	L	berechnet	Abweichung
00	65.788	23	61.824	+6,4%
01	171.153	64	172.032	-0,5%

Tabelle 3.19: Verkehrsaufkommen Cache Digests an lokale Cache-Server

Verglichen mit der Betrachtung des ICP-Verkehrs werden die gemessenen Werte für die übertragenen Cache Digests mit einer wesentlich geringeren Abweichung rechnerisch bestätigt. Die höhere Abweichung im Zeitraum 00 lässt sich zusätzlich durch die experimentelle Nutzung erklären.

Für die Berechnung von $R_{Among_{\text{Cache Digests}}}$ kann Gleichung 3.10 herangezogen werden. Dabei ist zu beachten, dass im Zeitraum 00 $n_1 = n_2 = 5$ und im Zeitraum 01 $n_1 = n_2 = 2$ gilt. In Tabelle 3.20 sollen zunächst die berechneten Werte von $R_{Among_{\text{Cache Digests}}}$ mit $f = 1$ betrachtet werden.

ID	gemessen	ohne Korrektur			mit Korrektur		
		f	berechnet	Abweichung	f	berechnet	Abweichung
00	27.782	1	26.880	+3,4%	1,0417	28.000	-0,8%
01	2.718	1	2.688	+1,1%	1,0119	2.720	-0,1%

Tabelle 3.20: Verkehrsaufkommen Cache Digests innerhalb DFN-Cache-Verbund

Die Abweichungen sind in beiden Zeiträumen gering, wobei in der Praxis Cache Digests häufiger übertragen wurden als aus den Berechnungen zu erwarten war. Die Ursache hierfür lag in den regelmäßigen Neustarts der DFN-Cache-Server. Ein Ca-

che-Server ruft unmittelbar nach jedem Neustart die Cache Digests der Neighbors ab, wodurch die Frequenz f erhöht wird. Im alten DFN-Cache-Verbund wurden täglich Neustarts von allen DFN-Cache-Servern durchgeführt, so dass die Cache Digests mit einer Frequenz von 25 statt ursprünglich 24 Cache Digests pro Tag abgerufen wurden. Dadurch ergibt sich eine korrigierte Frequenz von $f_{00} = 25/24 = 1{,}0417$ Cache Digests pro Stunde. Im neuen DFN-Cache-Verbund wurden die DFN-Cache-Server lediglich an zwei Tagen in der Woche neu gestartet, so dass sich die mittlere Frequenz auf $f_{01} = (24 \cdot 7 + 2)/(24 \cdot 7) = 1{,}0119$ Cache Digests pro Stunde erhöhte. Die korrigierten Werte sind ebenfalls in Tabelle 3.20 aufgeführt. Es zeigt sich, dass in beiden Zeiträumen die gemessenen Werte bei Abweichungen von unter 1% sehr gut mit den berechneten übereinstimmen.

3.3.2.4 Zusammenfassung

In Abbildung 3.8 sind die Werte aller in den vorangegangenen Tabellen aufgeführten Verkehrsflüsse grafisch zusammengestellt. Aus dieser Darstellung lassen sich sämtliche Entwicklungen ablesen. So zeigt sich, dass die Entwicklung der Requests wesentlich durch den ICP-Verkehr bestimmt wurde. Mit der Reduzierung von internem ICP-Verkehr $Among_{\mathrm{ICP}}$ (Datensätze 00 und 01) und externem ICP-Verkehr Out_{ICP} (Datensatz 01) konnte bei gleichzeitigem Anstieg der übertragenen Nutzdaten die Anzahl der bearbeiteten Requests mehr als halbiert werden.

Demgegenüber zeigt die Entwicklung des übertragenen Datenvolumens einen kontinuierlichen Anstieg. Gleichzeitig ist zu erkennen, dass das gesamte Datenvolumen der Inter-Cache-Kommunikation, verglichen mit den übertragenen Nutzdaten, nicht signifikant ist. Lediglich aus der Übertragung der Cache Digests an die lokalen Cache-Server ergibt sich ein erkennbarer Anteil ($Out_{\mathrm{Cache\ Digests}}$, Datensatz 01).

Der Verlauf der Übertragungsdauer gleicht dem der Requests, obwohl sich hier andere Ursachen anführen lassen. Ähnlich dem übertragenen Datenvolumen wird auch die Übertragungsdauer von den Nutzdaten und nicht von der Inter-Cache-Kommunikation bestimmt. Dass die gesamte Übertragungsdauer dennoch sank, ist wesentlich in den bereits angesprochenen externen Faktoren, wie der Verbesserung der Kernnetze und Netzübergänge, begründet. Weiterhin müssen auch der Umstieg auf das Betriebssystem Solaris 2.7 mit einem verbesserten TCP/IP-Stack sowie der Einsatz einer optimierten Squid-Version als Ursachen für die reduzierte Übertragungsdauer angeführt werden.

Insgesamt ist bei der Betrachtung von Abbildung 3.8 hervorzuheben, dass die kumulierten Übertragungszeiten über die letzten drei Zeiträume (Datensätze 99 bis 01) bei gleichzeitiger Verdoppelung des übertragenen Datenvolumens mehr als halbiert werden konnten.

Der Vergleich der gemessenen und berechneten Werte für die auftretenden Requests zur Inter-Cache-Kommunikation zeigt, dass die hergeleiteten Gleichungen nur unter

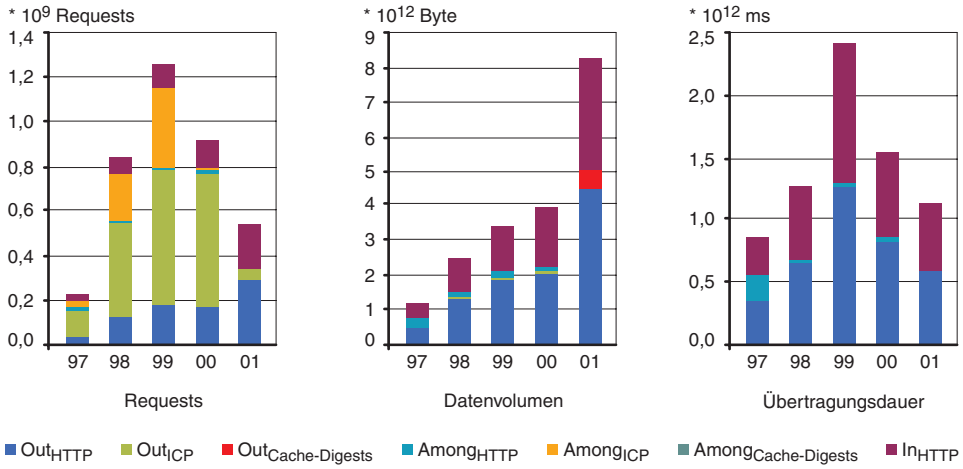


Abbildung 3.8: Verteilung der Verkehrsflüsse

sorgfältiger Beachtung der Randbedingungen anwendbar sind. Werden diese Bedingungen erfüllt, ergeben sich jedoch sehr gut Näherungen.

3.3.3 Objekt- und Volumen-Trefferraten

Die Berechnung der Objekt- und Volumen-Trefferraten unter ausschließlicher Berücksichtigung der übertragenen Nutzdaten ergibt sich aus den Gleichungen in Kapitel 2.2.2.1 und den Werten aus Tabelle 3.14. Für eine korrekte Berechnung der tatsächlichen Einsparungen durch einen Cache-Verbund müssen jedoch die zusätzlichen Verkehrsflüsse in geeigneter Weise einbezogen werden. Da hierfür keine allgemein geltenden Formeln bekannt sind, wird im Folgenden ein mögliches Vorgehen zur Berechnung der Volumen-Trefferrate vorgestellt.

Ein Vergleich von Abbildung 2.7 und Abbildung 3.6 zeigt, dass das Volumen der übertragenen Nutzdaten Out_{WWW} um das Volumen der Inter-Cache-Kommunikation Out_{ICC} zu verringern ist. Zusätzlich müssen die Verkehrsflüsse $Among_{WWW}$ und $Among_{ICC}$ innerhalb des Cache-Verbundes berücksichtigt werden. Da diese Verkehrsflüsse nur bei eingehenden Requests auftreten, die keinen unbestätigten Cache-Hit erzeugen, werden sie zu den eingehenden Nutzdaten In_{WWW} addiert.

Die Verkehrsflüsse innerhalb des Cache-Verbundes und die Inter-Cache-Kommunikation mit den Klienten werden nicht mit derselben Wertigkeit wie die eigentliche Übertragung der Nutzdaten berücksichtigt. Durch Einfügen eines Kostenfaktors k mit $0 \leq k \leq 1$ lässt sich in der Berechnung die Bedeutung dieser Verkehrsflüsse steuern. Ausgehend von der Berechnung der Volumen-Trefferrate eines einzelnen Cache-Servers in Gleichung 2.2 ergibt sich aus den genannten Erweiterungen Gleichung 3.11 für einen Cache-Verbund.

$$H_{V_k} = 1 - \frac{V_{In_{WWW}} + k \cdot (V_{Among_{WWW}} + V_{Among_{ICP}} + V_{Among_{Cache-Digests}})}{V_{Out_{WWW}} - k \cdot (V_{Out_{ICP}} + V_{Out_{Cache-Digests}})} \quad (3.11)$$

Eine entsprechende Berechnung der Objekt-Trefferraten unter Berücksichtigung der gesamten Inter-Cache-Kommunikation erscheint nicht sinnvoll, da der Aufwand zur Verarbeitung von ICP-Verkehr wesentlich geringer zu bewerten ist als der von HTTP-Verkehr. Weiterhin zeigt ein Vergleich von Tabelle 3.14 und Tabelle 3.18, dass der Anteil an Requests zur Übertragung von Cache Digests nicht signifikant ist, so dass lediglich der Verkehrsfluss $Among_{WWW}$ in die Berechnungen einfließen sollte. Aus Gleichung 3.11 ergibt sich mit den genannten Vereinfachungen Gleichung 3.12 zur Berechnung der Objekt-Hitraten in einem Cache-Verbund.

$$H_{R_k} = 1 - \frac{R_{In_{WWW}} + k \cdot R_{Among_{WWW}}}{R_{Out_{WWW}}} \quad (3.12)$$

Die exakte Festlegung des Kostenfaktors k hängt von verschiedenen Parametern ab, die im Rahmen dieser Arbeit nicht weiter definiert werden. Unabhängig davon kann jedoch die Bedeutung des Kostenfaktors aus Tabelle 3.21 abgelesen werden. Neben den Trefferraten H_R und H_V unter ausschließlicher Beachtung der Nutzdaten werden auch die Trefferraten nach Gleichung 3.11 und Gleichung 3.12 für verschiedene Kostenfaktoren k dargestellt.

ID	Objekt-Trefferrate				Volumen-Trefferrate			
	H_R	$H_{R_{0,1}}$	$H_{R_{0,2}}$	$H_{R_{0,3}}$	H_V	$H_{V_{0,1}}$	$H_{V_{0,2}}$	$H_{V_{0,3}}$
97	21,91%	16,54%	11,16%	5,78%	19,05%	13,18%	7,29%	1,38%
98	39,72%	38,82%	37,93%	37,03%	33,43%	32,25%	31,07%	29,88%
99	39,50%	38,57%	37,65%	36,72%	32,11%	30,93%	29,74%	28,55%
00	31,74%	30,45%	29,17%	27,89%	19,09%	18,17%	17,25%	16,32%
01	33,78%	33,72%	33,66%	33,60%	30,37%	29,45%	28,50%	27,53%

Tabelle 3.21: Objekt- und Volumen-Trefferraten im DFN-Cache-Verbund

Obwohl der eigentliche Wert von k nicht näher definiert wurde, lassen sich aus den Veränderungen der Trefferraten verschiedene Erkenntnisse ableiten. So zeigt der alte DFN-Cache-Verbund mit der ursprünglichen Konzeption (Datensatz 97) eine deutliche Verringerung der Trefferraten bei steigendem Kostenfaktor. Hier ist ein deutlicher Hinweis auf die Mängel der Konzeption zu sehen, die zu einem unverhältnismäßig hohen Verkehrsaufkommen innerhalb des DFN-Cache-Verbundes führten. In den restlichen Datensätzen zeigt sich ein robustes Verhalten gegenüber k,

wodurch nachträglich die Verbesserungen durch die veränderten Konzeptionen in altem und neuem DFN-Cache-Verbund bestätigt werden.

3.3.4 Analyse des WWW-Verkehrs

Eine detaillierte Analyse des Verkehrsflusses Out_{WWW} liefert Informationen über die Entwicklung und die charakteristischen Eigenschaften des Verkehrs im World Wide Web. Neben Aussagen über allgemeine Trends fallen dabei auch Ergebnisse an, die für die Konfiguration eines Cache-Verbundes von zentraler Bedeutung sind. So führt die Aufschlüsselung des WWW-Verkehrs nach Toplevel Domains in Kapitel 3.3.4.4 zu den Informationen, die bei der domain-basierten Partitionierung eines Cache-Verbundes eine optimale Verteilung der Verkehrsflüsse gewährleisten. Weitere Untersuchungen des Verkehrsflusses Out_{WWW} im Zeitraum 01 werden in Kapitel 4.1 mit dem Ziel durchgeführt, Eingangslasten für das Simulationsmodell eines Cache-Verbundes zu definieren.

Die im Folgenden aufgeführten Untersuchungen liefern eine Übersicht darüber, nach welchen Parametern WWW-Verkehr untersucht werden kann und wie sich die Entwicklung des WWW-Verkehrs über die betrachteten Zeiträume vollzog. Die Darstellungen wurden so gewählt, dass die qualitativen Abhängigkeiten und Auffälligkeiten erkannt werden, die bei der Modellierung zu berücksichtigen sind.

3.3.4.1 Kommunikationsbeziehungen

Die in Abbildung 3.9 dargestellten Verteilungen belegen erneut, dass in der ursprünglichen Konzeption (Datensatz 97) ein erheblicher Anteil des HTTP-Verkehrs innerhalb des DFN-Cache-Verbundes durch die Parent-Beziehungen zwischen den DFN-Cache-Servern verursacht wurde. Der Anteil von Neighbor-Misses und Neighbor-Hits am gesamten Verkehrsaufkommen überstieg sowohl bezüglich Requests, Datenvolumen und Übertragungsdauer 60%.

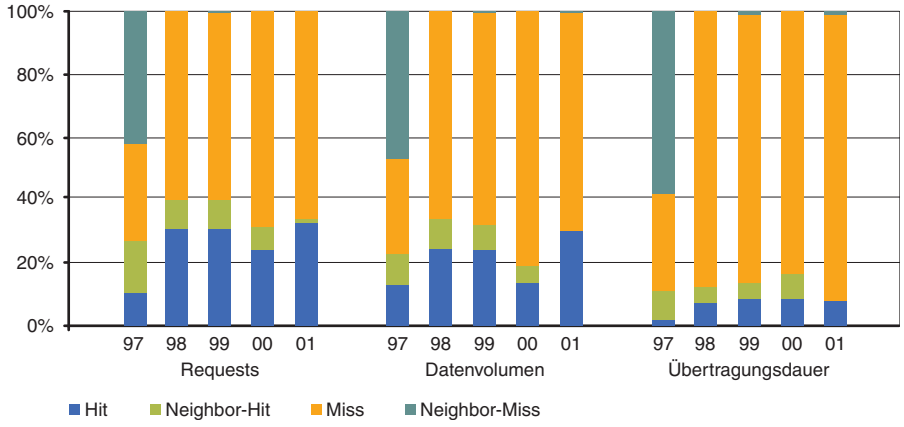


Abbildung 3.9: Verteilung von Cache-Hits und Cache-Misses

Mit dem Einsatz von Sibling-Beziehungen in der neuen Konzeption konnten die Neighbor-Misses nahezu vollständig reduziert werden, die verbleibenden Anteile wurden ausschließlich durch falsche Cache-Hits verursacht. Weiterhin zeigt sich, dass der nach Kapitel 2.3.3.3 bei Verwendung von Cache Digests zu erwartende Anstieg falscher Cache-Hits, der zu einem erhöhten Anteil von Neighbor-Misses in den Datensätzen 00 und 01 führen müsste, nicht zu erkennen ist. Daraus folgt, dass das Auftreten falscher Cache-Hits zu keiner signifikanten Beeinflussung der Verkehrsflüsse führt und als Argument gegen den Einsatz von Cache Digests nicht haltbar ist.

Da das hohe Aufkommen an Neighbor-Misses im Datensatz 97 die ausschlaggebende Ursache für Änderung der Konzeption war und bei der Planung von Cache-Verbünden weitgehend verhindert werden sollte, werden Neighbor-Misses in den folgenden Auswertungen das DFN-Cache-Verbundes noch berücksichtigt, bei der Modellierung in Kapitel 4 jedoch nicht betrachtet.

3.3.4.2 Statuscodes

Eine erste Klassifizierung der HTTP-Responses kann anhand der enthaltenen Statuscodes vorgenommen werden. Mit dem Statuscode signalisiert ein WWW-Server die erfolgreiche Bearbeitung des zugehörigen HTTP-Request oder liefert bei auftretenden Fehlern Informationen über die vermutete Ursache (s. Kapitel 2.1.3.2). Nach einer ersten Untersuchung konnten sieben verschiedene Statuscodes identifiziert werden, die signifikante Anteile an dem gesamten Verkehrsaufkommen verursachten (Abbildung 3.10).

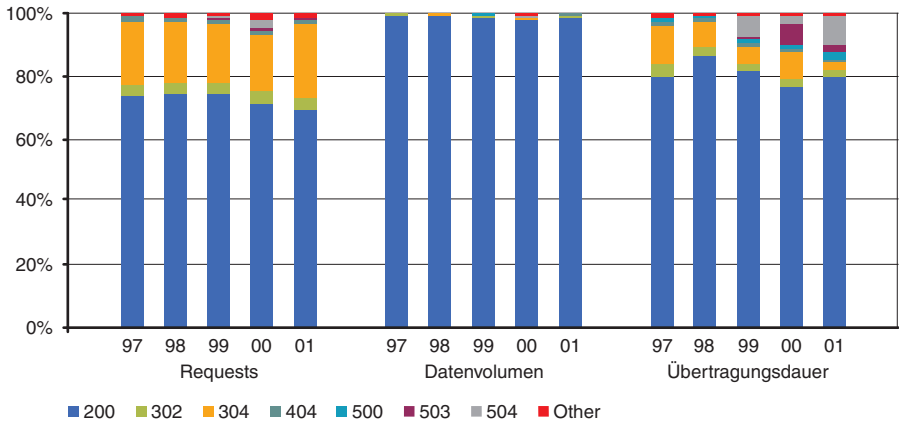


Abbildung 3.10: Verteilung der Statuscodes in HTTP-Responses

Zu erwarten waren die hohen Anteile folgender Statuscodes:

- `200` für eine erfolgreiche Übertragung (`OK`),
- `304` für bestätigte Cache-Hits (`Not Modified`),
- `302` für die Angabe alternativer URLs (`Moved Temporarily` bzw. `Found` nach HTTP/1.1),
- `404` für nicht vorhandene Objekte bzw. fehlerhafte URLs (`Not Found`).

Die nähere Betrachtung der Übertragungsdauer zeigt, dass drei weitere Statuscodes zu berücksichtigen sind. Die Statuscodes aus dem Bereich `5xx` signalisieren Fehler, die durch den WWW-Server verursacht werden. Neben dem Statuscode `500` (`Internal Server Error`) für eine nicht näher spezifizierbare Fehlermeldung kennzeichnet `503` (`Service Unavailable`) einen WWW-Server, der zwar HTTP-Requests empfängt und HTTP-Responses aussendet, das geforderte Objekt jedoch nicht in angemessener Zeit generieren oder aus dem Dateisystem auslesen kann. Ursache für diese Fehler sind häufig überlastete Datenbank-Anbindungen oder Festplatten-Systeme. Der mit HTTP/1.1 eingeführte Statuscode `504` (`Gateway Timeout`) wird von Cache-Servern generiert, wenn ein übergeordneter Server nicht erreichbar ist. Hierzu zählen sowohl übergeordnete Cache-Server als auch Netzkomponenten, die z. B. zur Last- oder Verkehrsverteilung in einer WWW-Server-Farm eingesetzt werden. Der Grund für die hohe Übertragungsdauer von HTTP-Responses mit diesen Statuscodes sind folglich nicht geringe Übertragungsraten, sondern Timeouts auf den Servern von mehreren Sekunden oder Minuten, nach deren Ablauf erst die HTTP-Responses generiert und übertragen werden.

Für die in Kapitel 3.3.4.3 bis Kapitel 3.3.4.7 vorgestellten Analysen werden ausschließlich die HTTP-Responses in Out_{WWW} betrachtet, die durch die Statuscodes 2xx oder 3xx eine erfolgreiche Bearbeitung des HTTP-Requests signalisieren. Dieses Verfahren ist in der Literatur üblich, um die Ergebnisse nicht durch die Einflüsse von z. B. fehlerhaften URLs und temporär gestörten Netzen oder Servern zu verfälschen. Dabei ist zu beachten, dass nach Abbildung 3.10 zwar über 90% der Requests und 95% des Datenvolumens erfasst werden, jedoch kaum mehr als 80% der gesamten Übertragungsdauer.

3.3.4.3 Dienste

Neben der Übertragung von HTTP-Verkehr können auch andere Dienste über einen Cache-Verbund abgerufen werden. Abbildung 3.11 zeigt die Anteile der verschiedenen Dienste, die vom DFN-Cache-Verbund bearbeitet wurden.

Die Darstellung belegt die überwiegende Bedeutung von HTTP, lediglich FTP-Verkehr weist einen weiteren erkennbaren Anteil am Verkehrsaufkommen auf. Dieser Anteil verringert sich jedoch kontinuierlich über den beobachteten Zeitraum. Bei der Interpretation ist zu beachten, dass über SSL verschlüsselter HTTP-Verkehr in der Regel bereits von lokalen Cache-Servern erkannt und unter Umgehung der DFN-Cache-Server direkt an die entsprechenden WWW-Server geleitet wurde. Dennoch zeigt sich, dass bei der Konzeption und Dimensionierung von einzelnen Cache-Servern oder Cache-Verbünden die Bearbeitung von WWW-Verkehr im Vordergrund stehen sollte.

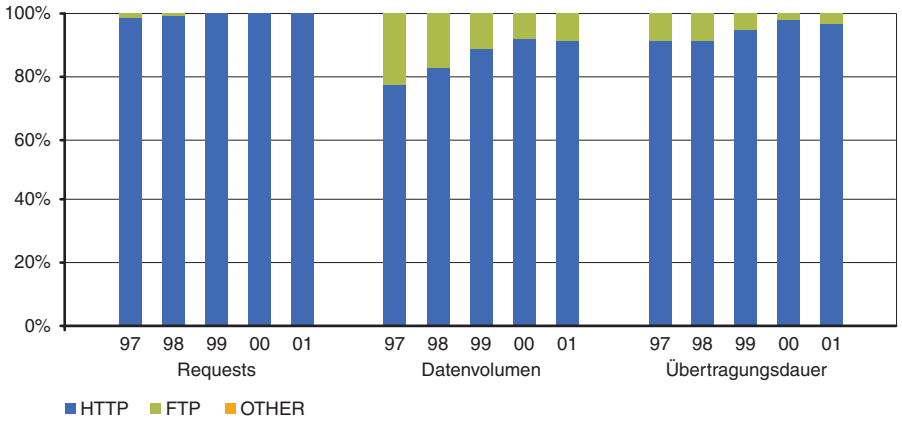


Abbildung 3.11: Abgerufene Dienste im DFN-Cache-Verbund

3.3.4.4 Toplevel Domains

Die Betrachtung des Verkehrsaufkommens nach verschiedenen Toplevel Domains ist eine notwendige Voraussetzung für die Bildung einer domain-basierten Partitionierung in einem Cache-Verbund. Erst aus der in Abbildung 3.12 dargestellten Statistik können die Anzahl und Größe der Partitionen sowie die geeigneten Mechanismen zur Inter-Cache-Kommunikation festgelegt werden.

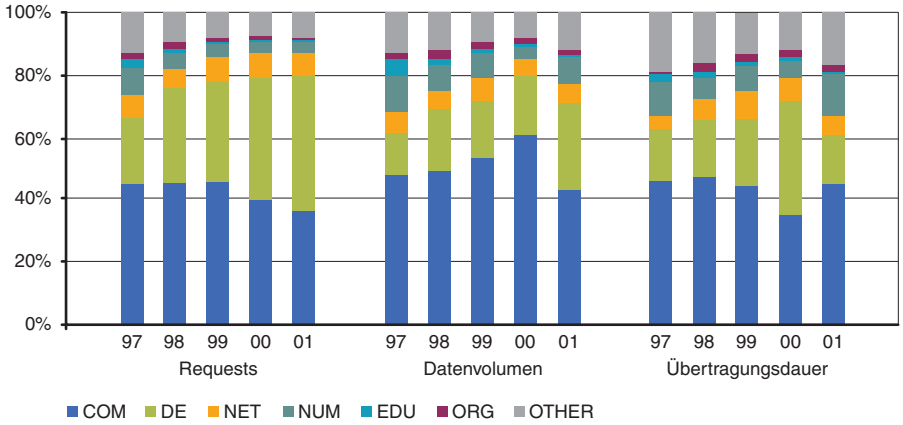


Abbildung 3.12: Verteilung nach Toplevel Domains

An dieser Stelle sei bereits auf einen Sachverhalt hingewiesen, der für die domain-basierte Partitionierung und die spätere Modellierung in Kapitel 4.1 von großer Bedeutung ist. Aus dem Anteil einer Toplevel Domain an den übertragenen Objekten kann nach Abbildung 3.12 nicht auf einen entsprechenden Anteil am Datenvolumen geschlossen werden. So beträgt im Datensatz 01 der Anteil der aus der Toplevel Domain `com` übertragenen Objekte 37%, das entsprechende Datenvolumen 43%. Demgegenüber beträgt der Anteil der Objekte aus der Toplevel Domain `de` 43%, was jedoch zu einem übertragenen Datenvolumen von lediglich 28% führte. Daraus folgt,

dass die Verteilungsfunktion der übertragenen Objektgrößen von der betrachteten Toplevel Domain abhängt. Dieser Tatsache ist besonders bei einer domain-basierten Partitionierung Rechnung zu tragen, da unterschiedliche Cache-Server bei gleicher Anzahl zu bearbeitender Requests unter Umständen deutlich verschiedene Datenvolumen übertragen.

Eine Bestätigung für die korrekte Auswahl der in Abbildung 3.12 betrachteten Toplevel Domains ergibt sich aus Abbildung 3.13. Ein so genanntes Ranking des Verkehrsaufkommens zeigt, dass die gewählten Toplevel Domains über den gesamten Projektzeitraum die höchsten Verkehrsaufkommen verursachten. Mit Ausnahme der Toplevel Domain `edu` nahmen die genannten Toplevel Domains stets die führenden Positionen sowohl bezüglich der Requests als auch des übertragenen Datenvolumens ein. Die Toplevel Domain `edu` verlor über den gesamten Projektzeitraum kontinuierlich an Bedeutung.

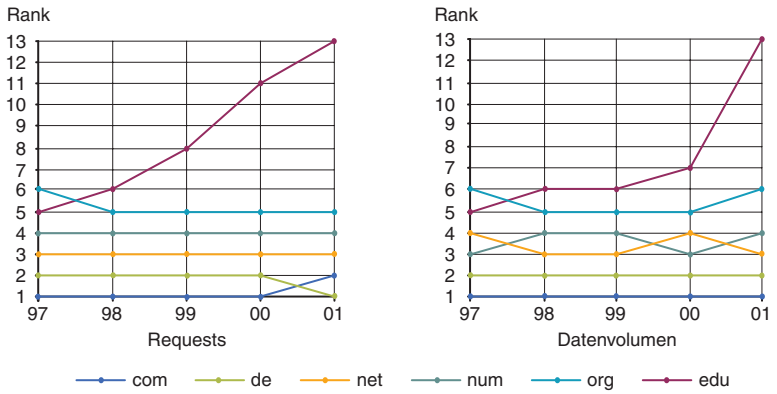


Abbildung 3.13: Ranking des Verkehrsaufkommens nach Toplevel Domains

3.3.4.5 Mimetypes

Nähere Informationen über die übertragenen Objekte liefert eine Betrachtung der Mimetypes. Der Mimetype eines Objekts wird im Allgemeinen aus der Endung des Dateinamens ermittelt. Dabei ist zu beachten, dass keine eindeutige Regelung zur Abbildung von Dateiendungen auf Mimetypes existiert [BoFr93]. So enthält der Entity-Header `Content-Type:` einer HTTP-Response einen Eintrag, den der WWW-Server einer textbasierten Konfigurationsdatei mit frei definierbaren Zuordnungen zwischen Dateiendungen und Mimetypes entnimmt. Für die im Rahmen der Arbeit durchgeführte Auswertung wurde zunächst die Angabe der Mimetypes in den Logdateien ausgewertet. Die Einträge wurden von der Cache-Software Squid direkt aus den genannten Entity-Headern der HTTP-Responses entnommen. Bei fehlender An-

gabe in den Logdateien wurde für die Untersuchung eine eigenständige Zuordnung nach Tabelle 3.22 vorgenommen.

Dateiendung	Mimetype
cab cac class dll doc dvi exe gz hqx jar js lha lzh pdf ppt ps rar rpm rtf swf tar tgz xls z zip	Application
aif aifc aiff au mid midi mp3 ra ram rm snd wav	Audio
bmp gif img ief jpe jpeg jpg pcx pgm pic png ppm ras rgb tif tiff xbm xpm xwd	Image
asp cfm css hbs htm html plain shtml sml map txt	Text
asf avi mpe mpeg mpg mov movie qt vdo viv vod	Video

Tabelle 3.22: Abbildung von Dateiendungen auf Mimetypes

Abbildung 3.14 zeigt die Verteilung der Mimetypes über die betrachteten Zeiträume. Auffällig ist der Unterschied zwischen den Anteilen an Requests und Volumen. In allen betrachteten Zeiträumen werden über 90% aller Requests, jedoch lediglich 40–50% des Datenvolumens durch Objekte der Mimetypes Image und Text verursacht. Das Datenvolumen wird wesentlich durch Objekte des Mimetypes Application bestimmt. Der Anteil von Objekten der Mimetypes Audio und Video beträgt auch am Datenvolumen weniger als 10%.

Weiterhin ist eine Korrelation in der Entwicklung des Datenvolumens und der Übertragungsdauer zu erkennen. Dennoch ist für eine detaillierte Modellierung zu klären, weshalb die jeweiligen Anteile signifikante Unterschiede aufweisen. So verursachen Objekte des Mimetypes Application im Zeitraum 01 50% des Datenvolumens und lediglich 30% der gesamten Übertragungsdauer. Dem stehen 30% des Datenvolumens für die Mimetypes Image und Text gegenüber, deren Anteil an der Übertragungsdauer jedoch 50% beträgt. Als Ursache dieser divergierenden Anteile ist der Einfluss von Trefferraten und die unterschiedliche Erreichbarkeit der jeweiligen WWW-Server zu vermuten.

Die dargestellte Auswertung der Mimetypes unterliegt der erläuterten, nicht eindeutigen Abbildung von Dateiendungen auf Mimetypes. Es ist bekannt, dass in einigen Fällen Objekte der Mimetypes Application, Audio und Video durch Anfügen von Endungen wie z. B. `.gif` an den Dateinamen als Mimetype Image maskiert werden. Ziel dieser Manipulation ist die Umgehung inhalt-basierter Filter, die z. B. lediglich die Übertragung von Objekten der Mimetypes Image und Text gestatten. Wie weit hiervon die dargestellten Ergebnisse beeinträchtigt werden, lässt sich aus den zur Verfügung stehenden Informationen nicht ermitteln. Im Folgenden wird vorausgesetzt, dass aufgrund der hohen Anzahl an ausgewerteten HTTP-Responses die Auswirkungen von Objekten mit nicht korrekten Mimetypes vernachlässigt werden können.

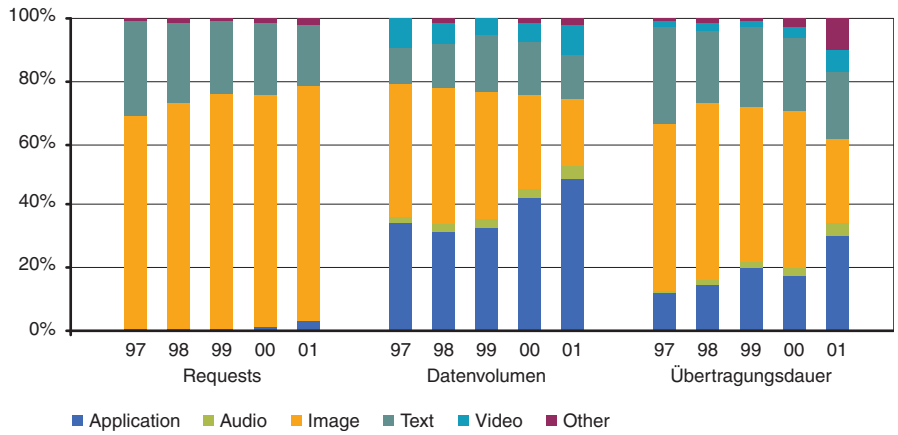


Abbildung 3.14: Verteilung der Mimetypes

3.3.4.6 WWW-Server

Die Anzahl der verschiedenen WWW-Server, von denen über den DFN-Cache-Verbund Informationen abgerufen wurden, liefert neben der Entwicklung der Verkehrsflüsse einen weiteren Hinweis auf das Wachstum des World Wide Web. Der in Tabelle 3.11 dargestellte Vergleich mit den im World Wide Web insgesamt verfügbaren WWW-Servern [Net02] zeigt, dass Anfang 1997 annähernd jeder vierte WWW-Server, Ende 2000 nicht einmal jeder 20. WWW-Server über den DFN-Cache-Verbund genutzt wurde.

ID	WWW-Server über DFN-Cache-Verbund	WWW-Server im Internet
97	182.855	786.419
98	358.412	2.002.703
99	454.148	3.111.223
00	630.052	6.582.257
01	970.599	25.681.583

Tabelle 3.23: Über DFN-Cache-Verbund abgerufene WWW-Server

Die grafische Darstellung in Abbildung 3.15 zeigt einen annähernd konstanten Rückgang dieses Verhältnisses. Ein Grund für diese Entwicklung liegt in der zunehmenden Anzahl von WWW-Servern, deren Inhalte nur für stark eingeschränkte Nutzerkreise von Interesse sind.

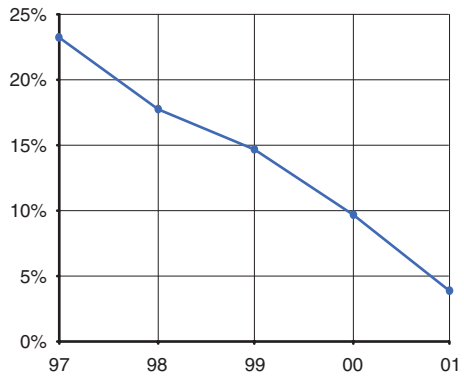


Abbildung 3.15: Über DFN-Cache-Verbund abgerufene WWW-Server

3.3.4.7 Popularität der übertragenen Objekte

Ergänzend zu der Betrachtung verschiedener WWW-Server in Kapitel 3.3.4.6 liefert eine Untersuchung der vollständigen URLs weitere Hinweise auf die steigende Anzahl und Vielfalt der verfügbaren Inhalte im World Wide Web. In Tabelle 3.24 werden die insgesamt übertragenen Objekte den verschiedenen Objekten sowie den One-Timern gegenübergestellt. Die aufgeführten One-Timer stellen die Objekte dar, die innerhalb des jeweils betrachteten Zeitraumes nur einmal über die DFN-Cache-Server abgerufen wurden. Da diese Objekte nicht mehrfach abgerufen wurden, lieferten sie keinen Beitrag zum Nutzen des DFN-Cache-Verbundes.

ID	Requests (aus Tabelle 3.14)	verschiedene Objekte (URLs)	One-Timer
97	30.103.738	10.348.969	6.359.310
98	115.838.801	22.943.254	13.417.349
99	168.729.518	34.876.252	20.731.284
00	151.300.244	38.295.275	24.868.244
01	268.102.647	56.825.017	38.160.775

Tabelle 3.24: Anzahl verschiedener Objekte im DFN-Cache-Verbund

Die Betrachtung der Häufigkeiten, mit denen einzelne Objekte abgerufen werden, liefert Anhaltspunkte für die Auswahl und die Implementierung von Ersetzungsstrategien auf einem Cache-Server. Die Verteilungsfunktion der so genannten Objekt-Popularität ist ein wesentlicher Parameter von Modellen, mit denen Trefferraten von Cache-Servern untersucht und simuliert werden (u. a. [BCF+98] [Pol01]).

Eine detaillierte Betrachtung der Objekt-Popularität ist für die Modellierung im Rahmen dieser Arbeit nicht erforderlich. Vielmehr soll mit Abbildung 3.16 lediglich

auf die Anteile der One-Timer an den verschiedenen Objekten sowie den insgesamt bearbeiteten Requests hingewiesen werden.

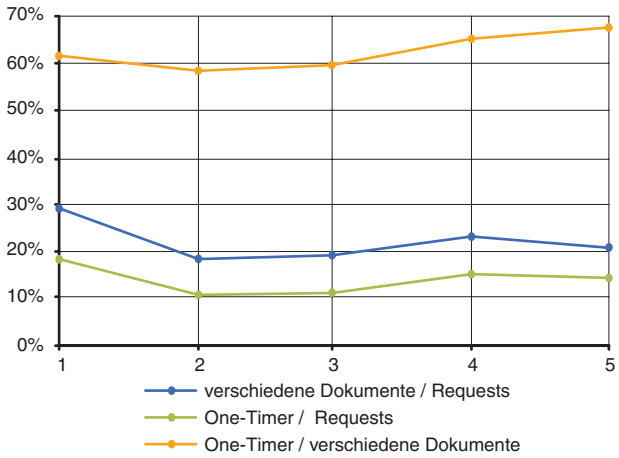


Abbildung 3.16: Anteile verschiedener Objekte und One-Timer

Der Anteil der One-Timer an den verschiedenen Objekten lag in allen Zeiträumen bei 60–65%. Der Nutzen des DFN-Cache-Verbundes ergibt sich aus den restlichen 35–40% der Objekte, die mehrfach abgerufen wurden. Demgegenüber lag der Anteil der One-Timer an allen abgerufenen Objekten lediglich bei ca. 15%. Dieser geringe Anteil zeigt, dass eine Vielzahl an Objekten zwei- oder mehrfach abgerufen wurde. Die Tatsache, dass beide Verhältnisse keinen signifikanten Schwankungen unterworfen waren, liefert eine Bestätigung für die annähernd konstanten Request-Trefferraten aus Tabelle 3.21.

Kapitel 4
Modellierung

Die Eingangslast in dem zu implementierenden Simulationsmodell wird durch die HTTP-Requests der lokalen Cache-Server repräsentiert. Sowohl die Inter-Cache-Kommunikation zwischen allen beteiligten Cache-Servern als auch die Übertragung von Nutzdaten innerhalb des Cache-Verbundes ergibt sich – in Abhängigkeit von der vorgegebenen Konfiguration – aus der Eingangslast. Um mit dem Simulationsmodell ein getreues Abbild der tatsächlichen Verkehrsflüsse in einem Cache-Verbund zu liefern, müssen daher die charakteristischen Eigenschaften des Verkehrsflusses Out_{WWW} aus Abbildung 3.6 möglichst genau nachgebildet werden.

Eine exakte Abbildung von Out_{WWW} wird durch eine so genannte trace-gesteuerte Simulation erreicht, bei der als Eingangslast direkt die Einträge aus den Logdateien eingelesen werden. Dieser Ansatz ist für die beabsichtigten Untersuchungen jedoch nicht geeignet. Zum einen müssten ständig sämtliche Logdateien mit einem gesamten komprimierten Volumen von über 80 GByte vorgehalten werden, zum anderen könnten Effekte bei variierenden Eingangslasten nicht untersucht werden. Darüber hinaus stellt die genaue Kenntnis der charakteristischen Eigenschaften von Out_{WWW} eine Voraussetzung für die korrekte Interpretation der aus den Simulationsdurchläufen erzielten Ergebnisse dar.

Daher wird der Implementierung des Simulationsmodells in Kapitel 5 eine ausführliche mathematische Modellierung von Out_{WWW} vorangestellt. Ziel der Modellierung ist eine Abbildung der gemessenen Verkehrscharakteristiken von Out_{WWW} durch mathematisch beschreibbare Funktionen. Da eine vollständige und korrekte Approximation in der Regel nicht möglich ist, wird ein Kompromiss zwischen möglichst genauer Abbildung und effizienter Implementierung der zugrunde liegenden Verteilungsfunktionen gewählt. Die hierfür notwendigen Verfahren und Algorithmen werden in den nächsten Kapiteln erläutert.

4.1 Vorgehen

4.1.1 Wahl der Verteilungsfunktionen

Die im Rahmen der Arbeit durchgeführten Messungen und Analysen der Verkehrsflüsse ergeben Zeitwerte, Dateigrößen oder Häufigkeiten. In den folgenden mathematischen Betrachtungen werden diese Ergebnisse durch eindimensionale kumulierte Wahrscheinlichkeitsverteilungen $F(x)$ einer Zufallsvariablen X mit $0 \leq F(x) \leq 1$ approximiert. Die Auswahl der geeigneten Verteilungsfunktionen, mit denen eine Approximation durchgeführt wird, orientiert sich an folgenden Aspekten:

- Die Funktionen müssen aus der Literatur bei der Untersuchung vergleichbarer Problemstellungen bekannt sein [Abd98] [ArWi97] [BBBC99] [CrBe97].
- Zur Bestimmung der Parameter müssen numerische Verfahren für lineare oder nichtlineare Regression existieren [PFTV90]. Diese Bedingung führt unter anderem dazu, dass ausschließlich zweiparametrige Verteilungsfunktionen betrachtet werden.
- Für die Implementierung des Simulationsmodells müssen die Verteilungen über generierende Funktionen entsprechender Zufallsfolgen, z. B. durch Berechnung der Invarianten, darstellbar sein [Jai91] [LaKe00].

Als geeignete Verteilungsfunktionen werden im Rahmen der Arbeit Exponential-, Lognormal-, Normal-, Pareto- und Weibull-Verteilung verwendet. Neben diesen kontinuierlichen Funktionen wird für die Betrachtung der Zwischenankunftszeiten ergänzend die Poisson-Verteilung herangezogen. Die genauen Formulierungen dieser Verteilungsfunktionen werden in Anhang B tabellarisch zusammengestellt. Neben der Wahrscheinlichkeitsdichte $f(x)$ und der kumulierten Verteilungsfunktion $F(x)$ werden auch die charakteristischen Maßzahlen Mittelwert, Standardabweichung, Median und Modalwert aufgeführt. Anhand der Maßzahlen kann die geeignete Verteilungsfunktion einfacher bestimmt und eine erste Abschätzung der Parameter bei der Regression vorgenommen werden.

4.1.1.1 Long-Tail-Verteilungsfunktionen

Eine besondere Klasse von Verteilungsfunktionen stellen die so genannten Long-Tail-Verteilungen (auch Fat- oder Heavy-Tail) dar. Diese Funktionen werden dadurch charakterisiert, dass ihre komplementär kumulativen Verteilungsfunktionen $\bar{F}(x) = 1 - F(x)$ für große x langsamer gegen 0 streben als die einfache Exponentialfunktion e^x. Eine Untermenge der Long-Tail-Funktionen bilden die Power-Tail-Funktionen, die sogar langsamer gegen 0 streben als die verallgemeinerte Exponentialfunktion $c \cdot a^x$ [BHW89]. In Gleichung 4.1 sind die verschiedenen Definitionen zusammengestellt.

$$
\begin{array}{lll}
\text{short tail} & \lim_{x\to\infty}(e^{ax}\cdot\bar{F}(x))\to 0 & a>0 \\
\text{exponential tail} & \lim_{x\to\infty}(e^{ax}\cdot\bar{F}(x))\to k & a>0,\ k>0 \\
\text{long tail} & \lim_{x\to\infty}(e^{ax}\cdot\bar{F}(x))\to\infty & a>0 \\
\text{power tail} & \lim_{x\to\infty}(a^{x}\cdot\ \bar{F}(x))\to k & a>0,\ k>0
\end{array}
\tag{4.1}
$$

Bereits die Betrachtung der charakteristischen Maßzahlen liefert Anhaltspunkte, ob eine empirische Häufigkeitsverteilung $\hat{F}(x)$ long-tail Eigenschaften aufweist. So deuten starke Unterschiede zwischen Median und Mittelwert oder eine hohe Standardabweichung auf eine zugrunde liegende Long-Tail-Verteilungsfunktion hin.

Verteilungsfunktionen mit long-tail Eigenschaften sind für eine analytische Modellierung ungeeignet, da in der Regel keine Laplace-Transformierte angegeben werden kann [FGSM01]. Um dennoch eine analytische Betrachtung von Systemen mit z. B. long-tail-verteilten Bedienzeiten durchführen zu können, wird über Hyperexponentialverteilungen eine Approximation mit Funktionen vorgenommen, deren Laplace-Transformierte existiert [FeWh98] [StSi00].

Da im Rahmen dieser Arbeit auf eine analytische Modellierung verzichtet wird (s. Kapitel 5.1.1), ist die Verwendung von Long-Tail-Verteilungsfunktionen ohne Einschränkung zulässig. Von den in Anhang B aufgeführten Funktionen weisen Lognormal- und Weibull-Verteilung long-tail Eigenschaften, die Pareto-Verteilung weist power-tail Eigenschaften auf.

4.1.1.2 Bestimmung der Parameter

Die Approximation einer theoretischen Verteilungsfunktion $F(x)$ an eine empirische Häufigkeitsverteilung $\hat{F}(x)$ erfolgt in zwei Schritten [Sch00]:

1. Bestimmung des geeigneten Verteilungstyps, z. B. durch Vergleich der gemessenen und berechneten charakteristischen Maßzahlen.
2. Schätzung der Parameter nach zwei unterschiedlichen Verfahren:
   - bei Exponential-, Pareto- und Weibull-Verteilung durch Approximation der linearisierten Wahrscheinlichkeitsverteilungen, z. B. mit der Methode der kleinsten Quadrate (Least Square Fitting).
   - bei Normal- und Lognormalverteilung z. B. durch Anwendung der Maximum-Likelihood-Schätzer (MLE).

Als Gütekriterium zur Bewertung der Parameter werden drei verschiedene Werte herangezogen [MCF00]:

$$
\begin{aligned}
D_{\mathrm{KS}} &= \lim_{0 \le x < \infty} \left| \mathrm{F}(x_i) - \hat{\mathrm{F}}(x_i) \right| && \text{maximaler Abstand, Kolmogoroff-Smirnov} \\
D_{\mathrm{A}} &= \frac{1}{N} \sum_{i=1}^{N} \left| \mathrm{F}(x_i) - \hat{\mathrm{F}}(x_i) \right| && \text{mittlerer Abstand} \\
D_{\mathrm{AQ}} &= \frac{1}{N} \sum_{i=1}^{N} \left(\mathrm{F}(x_i) - \hat{\mathrm{F}}(x_i) \right)^2 && \text{mittleres Abstandsquadrat}
\end{aligned}
\tag{4.2}
$$

Da $\mathrm{F}(x)$ und $\hat{\mathrm{F}}(x)$ einen Wertebereich von 0 bis 1 annehmen, gilt für die drei aufgeführten Gütekriterien $0 \le D \le 1$. Als Randbedingung für eine geeignete Approximation wird im Rahmen der Arbeit $D_{\mathrm{KS}} \le 0{,}1$ gefordert, d. h. es wird eine maximale Abweichung von 0,1 gegenüber dem originalen Wert zugelassen. Die weitere Auswahl der besten Approximation richtet sich nach dem kleinsten Wert von D_{A}.

Aufgrund des hohen Umfangs sowie der großen Spannweite der Stichproben war eine Bestimmung der Parameter über Statistikprogramme wie SPSS oder SAS nicht durchführbar. Daher wurden numerische Algorithmen aus [PFTV90] übernommen und auf die speziellen Anforderungen im Rahmen dieser Arbeit angepasst. Die vollständigen Ergebnisse der Approximationen werden unter Angabe der Gütekriterien in Anhang C aufgeführt.

4.2 Modellierung der Eingangslast

Die Qualität und Aussagekraft der mit dem Simulationsmodell erzielbaren Ergebnisse hängt wesentlich von der Genauigkeit ab, mit der die Eingangslast Out_{WWW} modelliert wird. Der Lastgenerator, dessen Eingangsparameter sich aus der Modellierung ergeben, stellt somit ein Modul von zentraler Bedeutung dar. Darüber hinaus liefert ein Vergleich zwischen den verschiedenen Datensätzen vertiefte Erkenntnisse über die Entwicklung des WWW-Verkehrs.

4.2.1 Vorüberlegungen

In der vorliegenden Problemstellung sind weder die speziellen Eigenschaften von TCP noch der detaillierte Protokollablauf von HTTP von Interesse. Vielmehr liegt der Schwerpunkt der Betrachtungen auf den charakteristischen Merkmalen der Objekte und Nachrichten, die über HTTP-Responses transportiert werden. Bei der Betrachtung von HTTP-Requests und -Responses ergibt sich theoretisch folgende Unterscheidung:

- Die HTTP-Requests der lokalen Cache-Server treffen auf den übergeordneten Cache-Servern in einer zeitlichen Abfolge ein, die durch die Zwischenankunftszeiten beschrieben wird. Da der Empfang und die Auswertung eines HTTP-Requests die gesamte Übertragungsdauer nur unwesentlich verlängert, kann der Zeitpunkt der Ankunft des HTTP-Requests mit der Beginn der Übertragung der resultierenden HTTP-Response gleichgesetzt werden. Eine nähere Betrachtung hierzu erfolgt mit dem Entwurf des Warteschlangenmodells in Kapitel 5.1.2.

- Aus der Vernachlässigung der HTTP-Requests folgt weiterhin, dass sich die Bedienzeiten im Simulationsmodell lediglich aus der Übertragungsdauer der resultierenden HTTP-Responses ergeben. Die Übertragungsdauer kann unmittelbar aus dem Quotient von Objektgröße und mittlerer Datenrate angegeben werden.

Eine vollständige Nachbildung des Datenstroms Out_{WWW} ergibt sich daher aus der Beschreibung der drei Merkmale Objektgröße, Datenrate und Zwischenankunftszeit. Die Ausführungen in Kapitel 3.3.4 zeigen, dass weder für die Objektgrößen noch für die Datenraten jeweils eine einzige Verteilungsfunktion angegeben werden kann, die sämtliche HTTP-Responses geeignet beschreibt. Vielmehr sind zunächst die Abhängigkeiten der drei Merkmale von den Parametern Mimetype, HTTP-Statuscode, Toplevel Domain sowie Cache-Hit bzw. Cache-Miss zu prüfen. Da die Datenraten und Zwischenankunftszeiten implizit zeitabhängige Merkmale sind, muss auch der Einfluss der gegenwärtigen Lastsituation bzw. der Tageszeit berücksichtigt werden.

Eine theoretisch mögliche vollständige Berücksichtigung aller dargestellten Abhängigkeiten würde zu einem hohen Aufwand sowohl bei der Modellierung als auch bei der Implementierung führen. Die Betrachtung von Cache-Hits bzw. Cache-Misses ergibt nach Kapitel 3.2.4 für HTTP-Responses mit den Statuscodes 200 und 304 insgesamt 18 verschiedene Kombinationen und jeweils vier verschiedene Kombinationen für die restlichen sechs Statuscodes. Mit den in Kapitel 3.3.4 aufgeführten sechs Mimetypes und sieben Toplevel Domains ergeben sich daraus $6 \cdot 7 \cdot (18 + 6 \cdot 4) = 1764$ Kombinationen, die noch in Abhängigkeit von der Tageszeit für jedes der drei Merkmale zu betrachten sind.

Die Merkmale Objektgröße, Datenrate und Zwischenankunftszeit umfassen einen breiten Ereignisraum und müssen durch kontinuierliche Verteilungsfunktionen approximiert werden. Demgegenüber können die Parameter ausschließlich über diskrete Verteilungsfunktionen mit weniger als zehn Elementarereignissen exakt beschrieben werden. Um die Anzahl der zu modellierenden Verteilungsfunktionen reduzieren zu können, müssen zunächst die Abhängigkeiten der Parameter untereinander untersucht und gegebenenfalls über diskrete Verteilungsfunktionen beschrieben werden. Daher werden der Modellierung zunächst Überlegungen vorangestellt, welche Abhängigkeiten zwischen Parametern und Merkmalen festgestellt werden können und welche vereinfachenden Annahmen zulässig sind.

4.2.2 Untersuchung der Abhängigkeiten

In einem ersten Schritt können aufgrund allgemein geltender Erfahrung unmittelbar verschiedene Abhängigkeiten ausgeschlossen werden. So wird die Datenrate nicht durch den Inhalt eines Objekts bestimmt und ist daher unabhängig von dem Mimetype. Die Objektgröße kann unabhängig von der Tageszeit betrachtet werden, wenn eine Übertragung ähnlicher Inhalte über den gesamten Tagesverlauf vorausgesetzt wird. Durch die hohe Anzahl und inhomogene Zusammensetzung der Nutzer im

Wissenschaftsnetz ist diese Voraussetzung erfüllt, so dass eine tageszeitabhängige Verteilung der Objektgrößen auszuschließen ist. Weiterhin werden die Inhalte einer HTTP-Response nicht aktiv durch Cache-Server verändert, so dass die Objektgröße auch von Cache-Hits bzw. Cache-Misses unabhängig ist.

4.2.2.1 Objektgrößen

Nachdem die Abhängigkeit der Objektgrößen von der Tageszeit sowie von Cache-Hits bzw. Cache-Misses ausgeschlossen werden kann, ist der Einfluss der restlichen Parameter zu prüfen. Aus der Literatur ist bekannt, dass eine Betrachtung nach verschiedenen Mimetypes zu unterschiedlichen Verteilungen der Objektgrößen führt (u. a. [Mah99] [JuCh99] [Bus00]). Diese Feststellung trifft nur für übertragene Nutzdaten zu, d. h. für HTTP-Responses mit den Statuscodes 200. HTTP-Responses mit Statuscodes ungleich 200 enthalten in der Regel textbasierte Nachrichten oder Fehlermeldungen, deren Größenverteilung nicht durch den Mimetype bestimmt wird.

Wie bereits anhand Abbildung 3.12 erläutert wurde, hängt die Größenverteilung der HTTP-Responses mit Statuscodes 200 auch von der betrachteten Toplevel Domain ab. Da diese Tatsache von zentraler Bedeutung für die domain-basierte Partitionierung in Cache-Verbünden ist, werden die restlichen Parameter in direkter oder indirekter Abhängigkeit von den Toplevel Domains betrachtet.

Um die Abhängigkeit der Objektgrößen von Mimetype und Toplevel Domain näher untersuchen zu können, wurden für den Datensatz 01 die charakteristischen Maßzahlen Median, Mittelwert und Standardabweichung der verschiedenen Verteilungen ermittelt. Den in Abbildung 4.1 dargestellten Ergebnissen ist zu entnehmen, dass die Maßzahlen eines Mimetypes bei verschiedenen Toplevel Domains nur geringen Schwankungen unterworfen sind. Daraus folgt, dass die Größenverteilung eines Mimetypes weitgehend unabhängig von der betrachteten Toplevel Domain ist. Die unterschiedlichen Anteile von Requests und Datenvolumen in Abbildung 3.12 sind offenbar dadurch zu erklären, dass die Anteile der Mimetypes zwischen den Toplevel Domains signifikant variieren. Diese Feststellung wird durch die Untersuchungen in Kapitel 4.2.6 bestätigt.

Zusammenfassend lassen sich folgende Abhängigkeiten feststellen, die bei der Modellierung der Objektgrößen und bei der Implementierung des Lastgenerators (vgl. Abbildung 4.3) zu berücksichtigen sind:

- Sämtliche Größenverteilungen der HTTP-Responses hängen direkt oder indirekt von der Toplevel Domain ab.
- Die Verteilung der Statuscodes in den HTTP-Responses hängt direkt von der Toplevel Domain ab.
- Es ist eine differenzierte Betrachtung zwischen HTTP-Responses mit den Statuscodes 200 und ungleich 200 erforderlich.

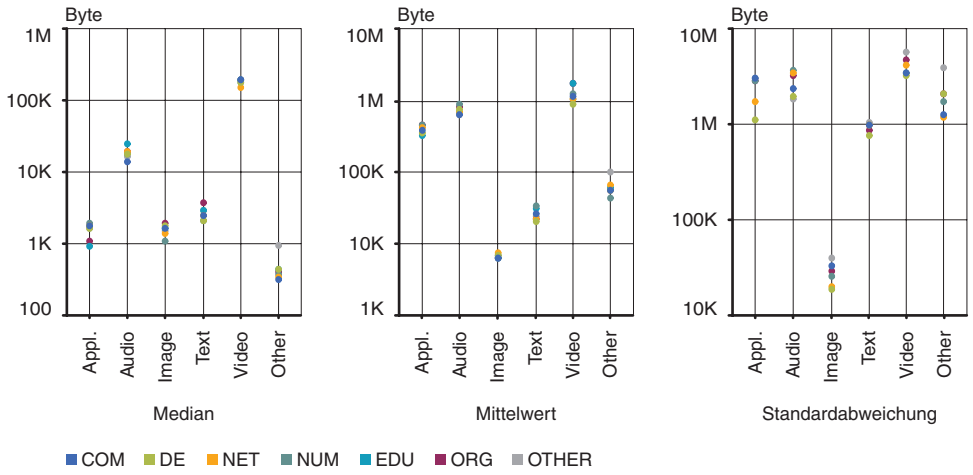


Abbildung 4.1: Maßzahlen in Abhängigkeit von Mimetype und Toplevel Domain

- Die Größenverteilung der HTTP-Responses mit den Statuscodes ungleich 200 hängt direkt von den Statuscodes ab.
- Die Größenverteilung der HTTP-Responses mit den Statuscodes 200 hängt direkt von den Mimetypes ab.
- Die Verteilung der Mimetypes hängt direkt von der Toplevel Domain ab.

4.2.2.2 Datenraten

Die Abhängigkeit der Datenraten von Cache-Hits bzw. Cache-Misses ist offensichtlich. Es ist zu erwarten, dass Cache-Server, die in einem nicht überlasteten Zustand betrieben werden, Cache-Hits mit deutlich höherer Übertragungsrate ausliefern als Cache-Misses.

Die Berücksichtigung der Toplevel Domains ist auch bei der Modellierung der Datenraten von hoher Bedeutung. So liefern die Datenraten von Cache-Hits Informationen über die Performance der jeweiligen Partition des Cache-Verbundes. Die Datenraten von Cache-Misses können als Indikator für die allgemeine Erreichbarkeit der WWW-Server aus der jeweiligen Toplevel Domain angesehen werden.

Nach Kapitel 3.2.3 und Kapitel 3.3.4.2 wird die Übertragungsdauer von HTTP-Responses mit den Statuscodes 504 maßgeblich durch festgelegte Timeouts bestimmt. Es bietet sich daher an, bei Cache-Misses aus der Verteilung der Übertragungsdauer dieser HTTP-Responses die Timeouts zu ermitteln und aus dem Quotient mit der Objektgröße die mittlere Datenrate zu berechnen. Dieses Vorgehen ist für alle Statuscodes ungleich 504 nicht anwendbar, d. h. in diesen Fällen muss die Datenrate über kontinuierliche Verteilungsfunktionen approximiert werden.

Neben den genannten Parametern werden die Datenraten maßgeblich von der aktuellen Lastsituation auf den Netzsegmenten und auf den Cache-Servern bestimmt. Da

die Lastsituation tageszeitbedingten Schwankungen unterworfen ist, muss die Betrachtung der Datenraten auf einen stationären Ankunftsprozess, d. h. auf ein Zeitfenster mit annähernd homogenem Verkehrsaufkommen, beschränkt werden. Um hohe Lastanforderungen geeignet betrachten zu können, sollte innerhalb des Zeitfensters ein hohes Verkehrsaufkommen vorliegen. Daher werden, wie bei der Dimensionierung von Telekommunikationsanlagen allgemein üblich, die Datenraten ausschließlich in der mittleren Hauptverkehrsstunde betrachtet. Durch die vor der Modellierung getroffene Festlegung wird eine tageszeitabhängige Betrachtung der Datenraten vermieden.

4.2.2.3 Zwischenankunftszeiten

Die Zwischenankunftszeiten der HTTP-Requests werden auf übergeordneten Cache-Servern nicht durch die übertragenen Inhalte bestimmt. Diese Annahme gilt nicht bei der Betrachtung lokaler Cache-Server, auf denen ein deutliches burstartiges Verkehrsaufkommen zu beobachten ist. Die Bursts werden, besonders bei der Verwendung von Multiple Simultaneous Connections (Kapitel 2.1.3.3), durch den Abruf einzelner WWW-Seiten mit einer Vielzahl eingebetteter Objekte und anschließenden Pausen während der Betrachtung der Inhalte verursacht [Mog95c] [CrBe97] [CCLS01]. Daraus folgt, dass die Zwischenankunftszeiten der HTTP-Requests auf lokalen Cache-Servern von der Zusammensetzung der WWW-Seiten und dem Verhalten der Nutzer abhängen. Eine geeignete Modellierung erfolgt in der Regel über On/Off-Modelle [BaCr98] [Cha00]. Demgegenüber sind auf übergeordneten Cache-Servern lediglich geringe Bursts von HTTP-Requests zu beobachten. Ursache hierfür sind die lokalen Cache-Server, die durch Cache-Hits die auftretenden Bursts nur in reduzierter Form weitergeben und durch Limitierung der eigenen Performance zu einer Glättung des Verkehrs beitragen [Che99] [RSGR00] [CCLS02].

Aus diesen Betrachtungen folgt, dass die Zwischenankunftszeiten der HTTP-Requests auf übergeordneten Cache-Servern ausschließlich von der betrachteten Lastsituation bzw. Tageszeit abhängen. Analog zu der Betrachtung der Datenraten werden auch die Zwischenankunftszeiten ausschließlich während der mittleren Hauptverkehrsstunde modelliert.

4.2.2.4 Zusammenfassung

In Abbildung 4.2 werden die in den Vorüberlegungen formulierten Abhängigkeiten zusammenfassend dargestellt.

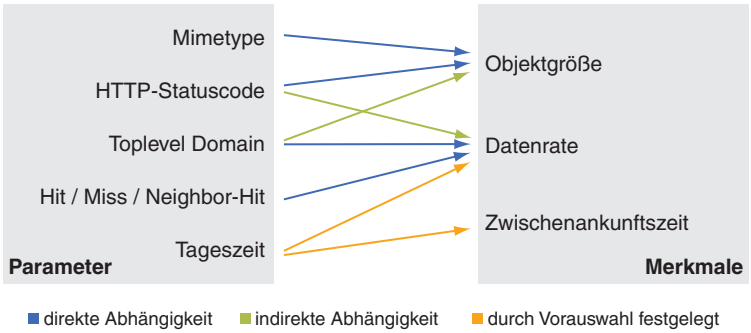


Abbildung 4.2: Abhängigkeiten zwischen Parametern und Merkmalen

Aus den in den vorangegangenen Abschnitten hergeleiteten Annahmen ergibt sich das in Abbildung 4.3 dargestellte Flussdiagramm zur Bestimmung von Datenvolumen und Datenrate einer HTTP-Response. Die grau unterlegten Felder kennzeichnen die Funktionen, für die bei der Modellierung eine Approximation über kontinuierliche Verteilungsfunktion durchgeführt werden muss. Das Flussdiagramm dient auch als Grundlage für die Implementierung des Lastgenerators in Kapitel 5.

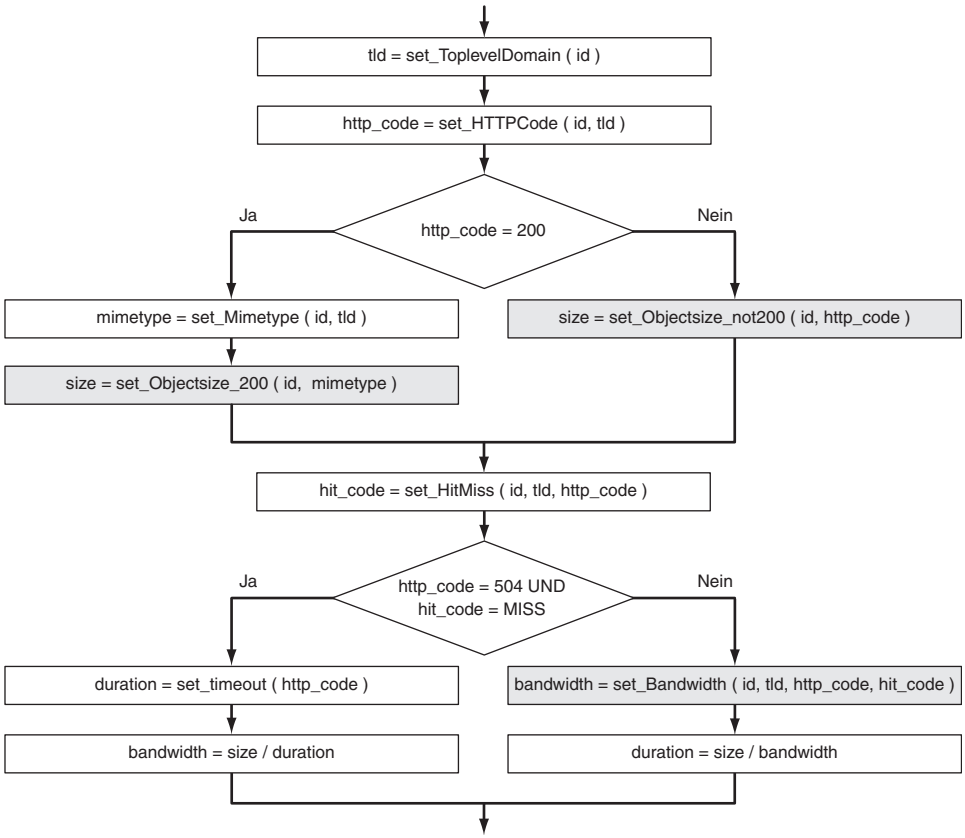


Abbildung 4.3: Flussdiagramm zur Bestimmung von Objektgröße und Datenrate

Mit Hilfe des Flussdiagramms kann die Anzahl der Verteilungen abgeleitet werden, für die nach den Vorüberlegungen noch eine Approximation durchzuführen ist. Die Herleitung hierfür wird unter Berücksichtigung der entsprechenden Parameter in Tabelle 4.1 dargestellt.

Merkmal	Statuscodes	Parameter	Anzahl	Gesamt
Objektgröße	200	Mimetype	6	6
	ungleich 200	Statuscode	8	8
Datenrate	200, 304	Toplevel Domain Hit / Miss	7 18	126
	304, 404, 500, 503, 504, 999	Toplevel Domain Statuscode Hit / Miss	7 6 4	168
Zwischenankunftszeit	alle	–	1	1
			Summe	309

Tabelle 4.1: Anzahl der durchzuführenden Approximationen

Gegenüber den ursprünglich angenommenen $3 \cdot 1764$ Verteilungsfunktionen konnte die Zahl auf 309 bzw. 6% gesenkt werden. Während die Anzahl der für die Objektgrößen zu betrachtenden Verteilungsfunktionen gering ist, ist eine vollständige Approximation von 294 verschiedenen Datenraten jedoch nicht handhabbar. Daher werden im Folgenden die Datenströme in dem Verkehrsfluss Out_{WWW} gesucht, die keinen signifikanten Anteil zu dem gesamten Verkehrsaufkommen beitragen und bei der Modellierung der Datenraten vernachlässigt werden können.

4.2.3 Reduzierung der zu betrachtenden Datenströme

Die Reduzierung der zu approximierenden Datenraten erfolgt in zwei Schritten. Im ersten Schritt werden die Datenströme identifiziert, die in dem betrachteten Verkehrsfluss Out_{WWW} nicht enthalten sind. Hierfür wird für Requests, Datenvolumen und Übertragungsdauer jeweils eine dreidimensionale Matrix erstellt, deren Achsen die Toplevel Domains, Statuscodes sowie Cache-Hits und Cache-Misses nach Tabelle 3.10 darstellen. Die Felder einer Matrix enthalten die jeweiligen prozentualen Anteile des Datenstroms bezogen auf das gesamte Verkehrsaufkommen in Out_{WWW}, so dass die Summe aller Einträge einer Matrix 100% ergibt. Aus dem Vergleich der Matrizen ergeben sich direkt die Datenströme, die nicht in Out_{WWW} enthalten sind, d. h. deren Anteile bezüglich Requests *und* Datenvolumen *und* Übertragungsdauer 0% betragen.

Mit dem zweiten Schritt erfolgt eine verlustbehaftete Reduzierung der Einträge in den Matrizen. Das bedeutet, dass Datenströme eliminiert werden, die zwar in Out_{WWW} auftreten, jedoch ohne wesentliche Beeinträchtigung der Ergebnisse vernachlässigt werden können. Für die Bestimmung der vernachlässigbaren Datenströ-

me wird eine untere Grenze k definiert. Sämtliche Felder, deren Werte in allen drei Matrizen gleichzeitig den Wert von k unterschreiten, werden auf den Wert 0% gesetzt und dadurch in Out_{WWW} eliminiert. Durch Variation von k ergibt sich die in Abbildung 4.4 dargestellte Reduzierung der in den unterschiedlichen Verkehrsflüssen verbleibenden Datenströme.

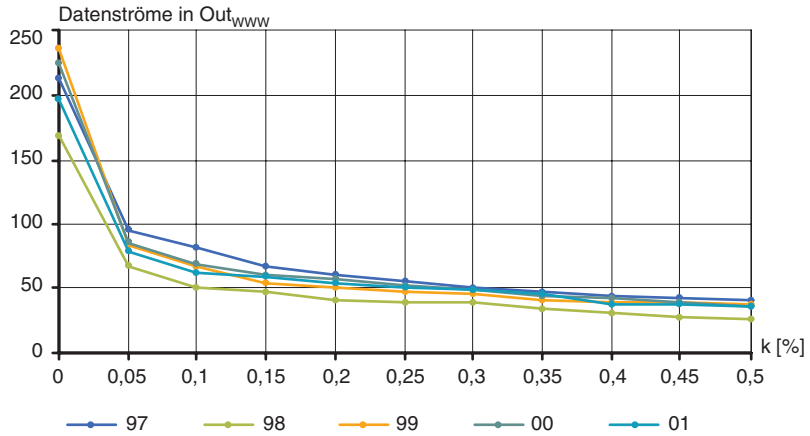


Abbildung 4.4: Verlustbehaftete Redzierung der Datenströme

Bereits mit Schritt 1 ($k=0\%$) wird gegenüber den ursprünglich 294 Datenströmen eine erkennbare Verringerung erzielt, wobei deutliche Unterschiede zwischen den Datensätzen sichtbar sind. Die Einführung der verlustbehafteten Reduzierung mit Schritt 2 führt bereits für $k=0{,}05\%$ zu einer signifikanten Verringerung, die weitere Erhöhung von k bis auf den Wert 0,5% liefert ein geringeres, annähernd lineares Absinken der verbleibenden Datenströme. Bei der Wahl eines geeigneten Wertes für k ist zu beachten, dass durch die verlustbehaftete Reduzierung die ursprünglichen Verkehrsflüsse in den betrachteten Datensätzen nicht signifikant verändert werden dürfen. Da Cache-Hits seltener und mit einer deutlich geringeren Übertragungsdauer auftreten als Cache-Misses, folgt daraus, dass die Auswirkungen der Vereinfachungen auf die Verkehrsflüsse In_{WWW} und $Out_{\mathrm{WWW}} - In_{\mathrm{WWW}}$ sowie die Hitraten differenziert zu prüfen sind.

In Abbildung 4.5 sind die entsprechenden Diagramme dargestellt, wobei, ausgehend von dem tatsächlichen Wert (entspricht $k=0\%$) für jeden Datensatz, die Merkmale mit steigendem Wert $0 \le k \le 0{,}5$ in Schritten von 0,05% betrachtet werden. Die Werte oberhalb von 100% in der Darstellung der Hitraten sind dadurch zu erklären, dass sich durch eine stärkere Verringerung von In_{WWW} gegenüber Out_{WWW} mit steigendem k höhere Objekt- und Volumen-Trefferraten ergeben. Weiterhin wurde in der Darstellung der Hitraten auch das entsprechende Verhältnis der Übertragungsdauer von In_{WWW} und Out_{WWW} angegeben. Dieser Wert ist ohne praktische Bedeutung und dient hier lediglich der vergleichenden Betrachtung.

Insbesondere anhand des Verkehrsflusses $Out_{\mathrm{WWW}} - In_{\mathrm{WWW}}$, d. h. für die Cache-Hits, wird die notwendige Sorgfalt bei der Wahl eines geeigneten Wertes für k deut-

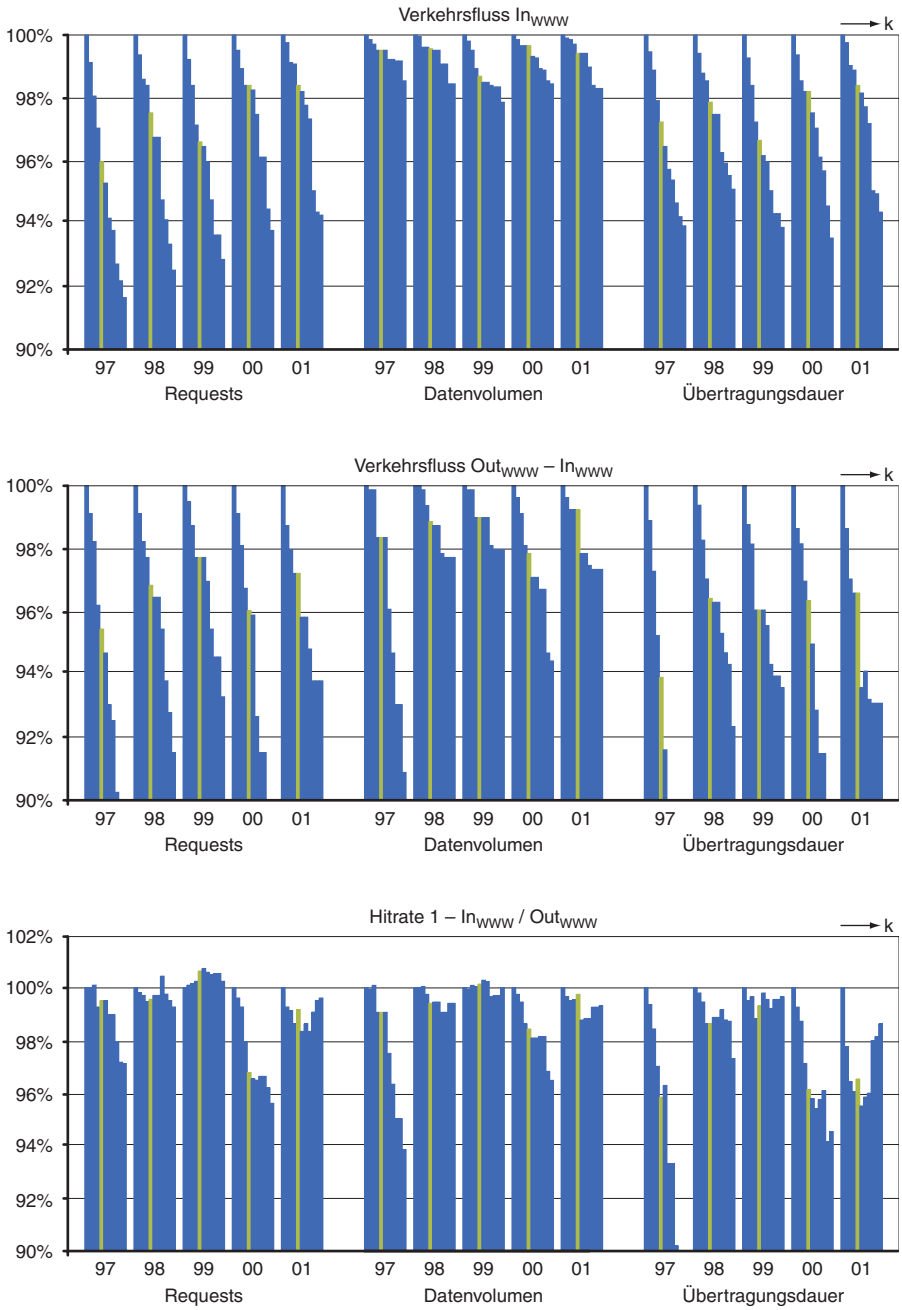


Abbildung 4.5: Verkehrsflüsse und Trefferraten für variierende k

lich. So führt eine Erhöhung von k um 0,05% teilweise zu einer Reduzierung der betrachteten Merkmale von über 2%. Als willkürliche Annahme für die Vereinfachung wurde festgelegt, dass jedes Merkmal mindestens durch 96% des ursprünglichen Wertes repräsentiert werden sollte. Aus dieser Forderung ergibt sich eine maximale untere Grenze von $k = 0{,}2\%$ für alle Datensätze. Die entsprechenden Werte der Merkmale für $k = 0{,}2\%$ sind in den Diagrammen grün markiert.

In Tabelle 4.2 werden die Datenflüsse für den Datensatz 01 dargestellt, die sich nach der Reduzierung ergeben. Zugunsten einer einfachen Darstellung wird hier lediglich die Matrix für die Requests angegeben, wobei die ursprünglich dreidimensionale Matrix auf eine zweidimensionale Tabelle abgebildet wird. Die Zeilen der Tabelle ergeben sich aus den Betrachtungen in Kapitel 3.2.4 für die Übertragung von Nutzdaten und Kontrollnachrichten. Die dunkelgrau hinterlegten Felder kennzeichnen die Datenströme, die in Schritt 1 identifiziert wurden. Entsprechend kennzeichnen die hellgrau hinterlegten Felder die Datenströme, deren Anteil bezüglich Requests, Datenvolumen und Übertragungsrate unterhalb von 0,2% des gesamten Verkehrsaufkommens in Out_{WWW} betrug.

Anhand der Markierungen ist unmittelbar zu erkennen, dass die wesentlichen Anteile in Out_{WWW} von den Hits und Misses der Nutzdaten sowie den Misses der Kontrollnachrichten verursacht werden. Sämtliche bei der Übertragung von Kontrollnachrichten anfallende Hits, Neighbor-Hits und Neighbor-Misses können unter den genannten Bedingungen vernachlässigt werden. Bei der Übertragung der Nutzdaten zeigt sich ebenfalls, dass Neighbor-Misses bis auf eine Ausnahme (`num`) und Neighbor-Hits bis auf zwei Ausnahmen (`com`, `de`) vollständig reduziert werden.

Die hier angeführten Beobachtungen lassen sich nur für den Datensatz 01 festhalten und können nicht auf die restlichen Datensätze übertragen werden. Das geringe Aufkommen an WWW-Verkehr zwischen den DFN-Cache-Servern im Datensatz 01 ist durch die zugrunde liegende Konzeption zu erklären. Nach der Konzeption im neuen DFN-Cache-Verbund (Kapitel 3.1.2.3) sollten lokale Cache-Server alle vier DFN-Caches als Parents nutzen, weshalb HTTP-Requests stets direkt an die DFN-Cache-Server mit den geforderten Objekten gesendet werden konnten.

Demgegenüber konnten lokale Cache-Server lediglich einen Teil der zehn DFN-Cache-Server im alten DFN-Cache-Verbund direkt nutzen, wodurch ein höheres Aufkommen an Neighbor-Hits in den restlichen Datensätzen zu erklären ist (Anhang C). Deutlich ist weiterhin der hohe Anteil von Neighbor-Misses in dem Datensatz 97. Der Grund hierfür liegt in der ursprünglichen Konzeption im alten DFN-Cache-Verbund, die Parent-Beziehungen zwischen den DFN-Cache-Servern vorsah (Kapitel 3.1.2.1 und Kapitel 3.2.4).

	Statuscode				HTTP-Responses (in %)						
	C_L-C_1	C_1-S	C_1-C_2	C_2-S	com	de	net	num	edu	org	other
Hit	200	–	–	–	7,86	8,82	1,60	0,61	0,00	0,20	1,30
	304	–	–	–	3,80	5,85	0,88	0,39	0,00	0,00	0,77
Miss	200	200	–	–	16,40	18,21	3,42	1,45	0,27	0,63	4,04
	200	304	–	–	1,22	1,69	0,29	0,00	0,00	0,00	0,33
	304	200	–	–	2,47	4,88	0,83	0,35	0,00	0,00	0,68
	304	304	–	–	0,70	1,31	0,00	0,00	0,00	0,00	0,27
Neighbor-Hit	200	–	200	–	0,25	0,30	0,00	0,00	0,00	0,00	0,00
	200	–	304	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	200	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	304	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Neighbor-Miss	200	–	200	200	0,00	0,00	0,00	0,02	0,00	0,00	0,00
	200	–	200	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	200	–	304	200	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	200	–	304	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	200	200	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	200	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	304	200	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	304	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Hit	302	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	404	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	500	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	503	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	504	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	999	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Miss	302	302	–	–	2,37	0,84	0,39	0,00	0,00	0,00	0,00
	404	404	–	–	0,37	0,44	0,00	0,00	0,00	0,00	0,11
	500	500	–	–	0,03	0,07	0,00	0,00	0,00	0,00	0,01
	503	503	–	–	0,03	0,01	0,00	0,01	0,00	0,00	0,00
	504	504	–	–	0,03	0,02	0,00	0,01	0,00	0,00	0,01
	999	999	–	–	0,77	0,39	0,00	0,00	0,00	0,00	0,00
Neighbor-Hit	302	–	302	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	404	–	404	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	500	–	500	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	503	–	503	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	504	–	504	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	999	–	999	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Neighbor-Miss	302	–	302	302	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	404	–	404	404	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	500	–	500	500	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	503	–	503	503	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	504	–	504	504	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	999	–	999	999	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Tabelle 4.2: Anteile HTTP-Responses nach der Reduzierung

Eine Übersicht über die mit den Vorüberlegungen und durchgeführten Vereinfachungen erzielte Reduzierung der zu betrachtenden Datenströme wird in Abbildung 4.6 zusammengefasst.

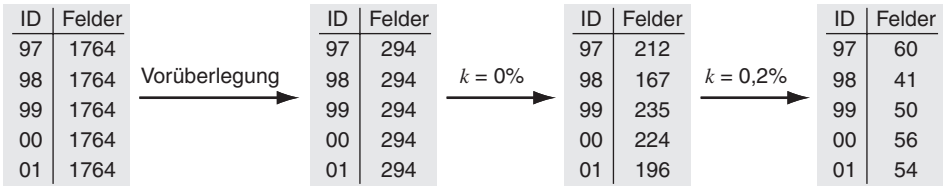


Abbildung 4.6: Übersicht Reduzierung der Datenströme

Ausgehend von den dargestellten Vereinfachungen werden in den folgenden Kapiteln die Merkmale Objektgröße, Datenrate und Zwischenankunftszeit in Abhängigkeit von den aufgeführten Parametern untersucht. Da das Vorgehen für alle betrachteten Datensätze identisch ist, werden die Berechnungen weitgehend nur für den Datensatz 01 beispielhaft dargestellt. Die Ergebnisse sämtlicher Datensätze werden in Anhang C tabellarisch zusammengestellt.

4.2.4 Toplevel Domains

Eine nach Toplevel Domains differenzierte Betrachtung der Merkmale wurde bereits in Abbildung 3.12 für alle Datensätze dargestellt. In Abbildung 4.7 werden lediglich die Verteilungen für den Datensatz 01 hervorgehoben. Die prozentuale Verteilung der Requests wird durch die Funktion `set_ToplevelDomain ( id )` in dem Lastgenerator abgebildet (s. Flussdiagramm in Abbildung 4.3). An dieser Stelle sei erneut auf die unterschiedlichen Anteile z. B. der Toplevel Domains `com` und `de` zwischen den Merkmalen hingewiesen, die durch den Lastgenerator nachzubilden sind.

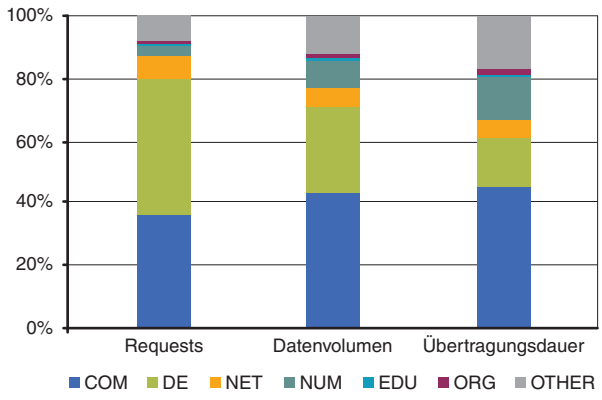


Abbildung 4.7: Verteilung Toplevel Domains

4.2.5 Statuscodes

Wie aus der Verteilung der Statuscodes in Abhängigkeit von der Toplevel Domain in Abbildung 4.8 zu entnehmen ist, beträgt der Anteil an HTTP-Responses mit Kontrollnachrichten stets ca. 30%. Während das Volumen dieser HTTP-Responses vernachlässigbar gering ist, werden auch ca. 20% der gesamten Übertragungsdauer für deren Übertragung aufgewendet. Hierdurch wird die Annahme bestätigt, dass eine Modellierung allein der übertragenen Nutzdaten zwar zu einer guten Beschreibung der Datenvolumen, jedoch zu einer ungenauen Nachbildung der Übertragungsraten führen würde. Die Verteilung der Statuscodes wird in dem Lastgenerator durch die Funktion `set_HTTPCode ( id, tld )` abgebildet.

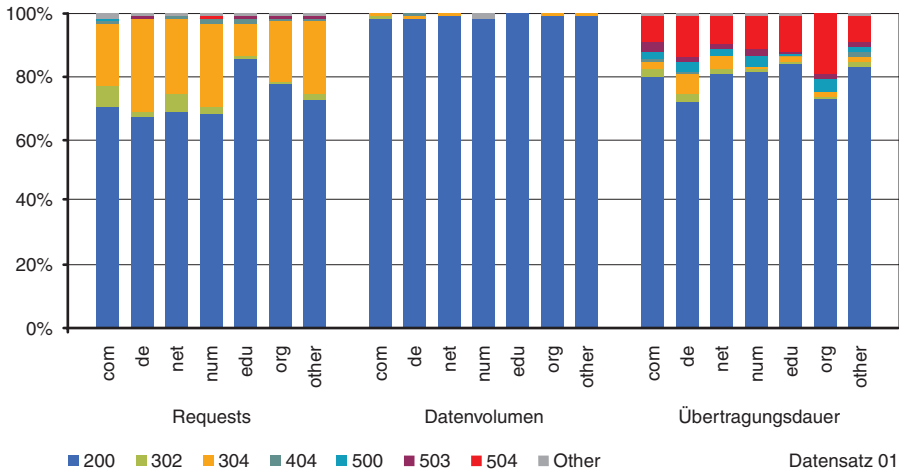


Abbildung 4.8: Verteilung HTTP-Codes in Abhängigkeit Toplevel Domains

4.2.6 Mimetypes

Die Abhängigkeit der Verteilung der Objektgrößen von Mimetypes und Toplevel Domains wurde bereits bei den Vorüberlegungen in Kapitel 4.2.2.1 näher betrachtet. Die in Abbildung 4.1 dargestellten Maßzahlen führten zu der Feststellung, dass die Größenverteilungen abhängig von den Mimetypes und unabhängig von den Toplevel Domains dargestellt werden können. Unabhängig davon sind jedoch die Häufigkeiten, mit denen Objekte verschiedener Mimetypes übertragen werden, getrennt nach Toplevel Domains zu untersuchen.

Die entsprechende Auswertung wird in Abbildung 4.9 dargestellt. Hier werden die unterschiedlichen Ausprägungen der Mimetypes zwischen den verschiedenen Toplevel Domains bestätigt. Zu beachten ist z. B., dass die erkennbaren Anteile von HTTP-Responses mit den Mimetypes Audio und Video in der Toplevel Domain `num` zu einer deutlichen Veränderung des übertragenen Datenvolumens führen. In dem hohen Anteil von Objekten mit großem Datenvolumen ist die Ursache für die in Abbildung 4.7 dargestellte Verteilung zu sehen, nach der die ca. 3% HTTP-Responses aus der Toplevel Domain `num` über 10% des übertragenen Datenvolumens

verursachen. Die Verteilung der Mimetypes wird in dem Generator durch die Funktion `set_Mimetype ( id, tld )` abgebildet.

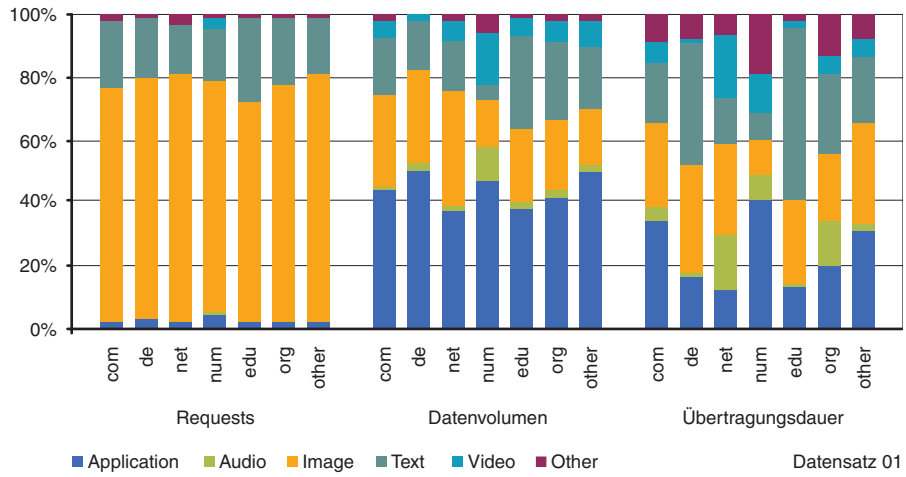


Abbildung 4.9: Verteilung Mimetypes in Abhängigkeit Toplevel Domains

4.2.7 Cache-Hits und Cache-Misses

Die Berechnung der Anteile von Cache-Hits und Cache-Misses wurde in Kapitel 4.2.3 dargestellt, die Werte können Tabelle 4.2 entnommen werden. Die Verteilung von Cache-Hits und Cache-Misses wird in dem Lastgenerator durch die Funktion `set_HitMiss ( id, tld, http_code )` abgebildet.

4.2.8 Objektgrößen

Die Modellierung der Objektgrößen erfolgt durch Approximation kontinuierlicher Verteilungsfunktionen. Nach Kapitel 4.2.2.1 ist dabei eine Unterscheidung zwischen HTTP-Responses mit Statuscodes 200 und den übrigen HTTP-Responses erforderlich.

4.2.8.1 Größenverteilung der Nutzdaten

Bei der Betrachtung der Größenverteilung von HTTP-Responses mit Statuscodes 200 ist eine abschnittsweise Approximation notwendig [Mah99]. Hierbei werden die gemessenen Verteilungen $\hat{F}(x)$ durch jeweils eine Verteilungsfunktion für den Body $F_b(x)$ und den Tail $F_t(x)$ approximiert. Die Notwendigkeit für dieses Vorgehen lässt sich anhand Abbildung 4.10 anschaulich darstellen.

In dem Beispiel wird die Größenverteilung des Mimetypes Text für den Datensatz 97 untersucht. Die Abbildung der CDF zeigt bereits, dass die Approximation der Requests durch lediglich eine Verteilungsfunktion das übertragene Datenvolumen nur unzureichend modelliert. Dem Vergleich von CDF und CCDF ist zu entnehmen, dass durch den Body ca. 95% der HTTP-Responses, jedoch lediglich 50% des Da-

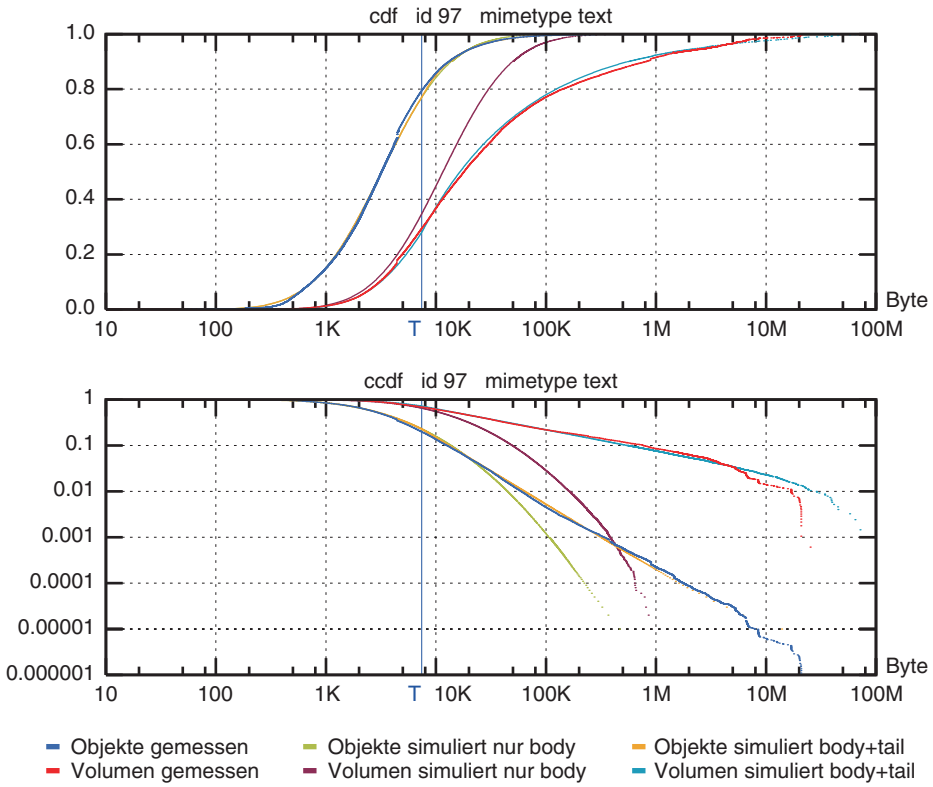


Abbildung 4.10: Approximation der Objektgröße über Body und Tail

tenvolumens in ausreichender Näherung modelliert werden. Daraus folgt unmittelbar, dass durch die verbleibenden 5% der Requests ebenfalls 50% des anfallenden Datenvolumens generiert werden. Erst mit der abschnittsweisen Modellierung über eine zusätzliche Verteilungsfunktion für den Tail wird auch das Datenvolumen in guter Näherung dargestellt.

In Abbildung 4.10 ist der Wert T (Transition) für den Übergang zwischen $F_b(x)$ und $F_t(x)$ markiert. Der Übergang zwischen $F_b(T)$ und $F_t(T)$ ergibt sich aus der Definition der Wahrscheinlichkeitsverteilung:

$$\int_0^\infty f(x)\,dx = F(\infty) - F(0) \equiv 1 \tag{4.3}$$

Mit der hier vorliegenden abschnittweisen Definition der Verteilungsfunktion

$$F(x) = \begin{cases} F_b(x) & 0 \le x < T \\ F_t(x) & T \le x < \infty \end{cases} \tag{4.4}$$

folgt daraus

$$\int_0^{\infty} f(x)\,dx = \int_0^{T} f_b(x)\,dx + \int_T^{\infty} f_t(x)\,dx = F_b(T) - F_b(0) + F_t(\infty) - F_t(T) = 1 \qquad (4.5)$$

Mit $F_b(0) = 0$ und $F_t(\infty) = 1$ ergibt sich für den Übergang zwischen Body und Tail die Forderung $F_b(T) = F_t(T)$.

Das hier verwendete Vorgehen bei der Modellierung der Größenverteilungen gliedert sich in fünf Schritte:

1. Zunächst wird nach dem in Kapitel 4.1.1.2 erläuterten Verfahren die Verteilungsfunktion $F_b(x)$ ermittelt, die den gesamten Verlauf der Requests modelliert.
2. Durch Simulation werden 1.000.000 Objekte mit einer Größenverteilung nach $F_b(x)$ generiert. Für diese Objekte wird die Verteilungsfunktion der Datenvolumen ermittelt.
3. Anhand der CCDF von gemessener und simulierter Größenverteilung der Datenvolumen wird der Übergang T ermittelt, ab dem keine ausreichende Näherung gegeben ist. Die Bestimmung von T richtet sich nach fest vorgegebenen Abweichungen oder aus der grafischen Darstellung der entsprechenden Kurven.
4. In dem Bereich $T \le x < \infty$ wird eine Approximation für $F_t(x)$ durchgeführt, wobei als Nebenbedingung $F_b(T) = F_t(T)$ zu beachten ist.
5. In einer weiteren Simulation werden 1.000.000 Objekte mit einer Größenverteilung nach $F_b(x) + F_t(x)$ generiert. Aus dem Vergleich der Verteilungsfunktion dieser Objekte mit den gemessenen Werten ergeben sich die Güteparameter D nach Kapitel 4.1.1.2.

Ergänzend ist anzumerken, dass zwischen den Wahrscheinlichkeitsverteilungen von Requests und Datenvolumen folgender Zusammenhang besteht, der sich bei der Berechnung der Verteilungsfunktionen jedoch nicht sinnvoll nutzen lässt:

$$F_{Req}(x) = \int_0^{\infty} f_{Req}(x)\,dx \quad \text{und} \quad F_{Vol}(x) = \int_0^{\infty} x \cdot f_{Req}(x)\,dx \qquad (4.6)$$

Ergebnisse

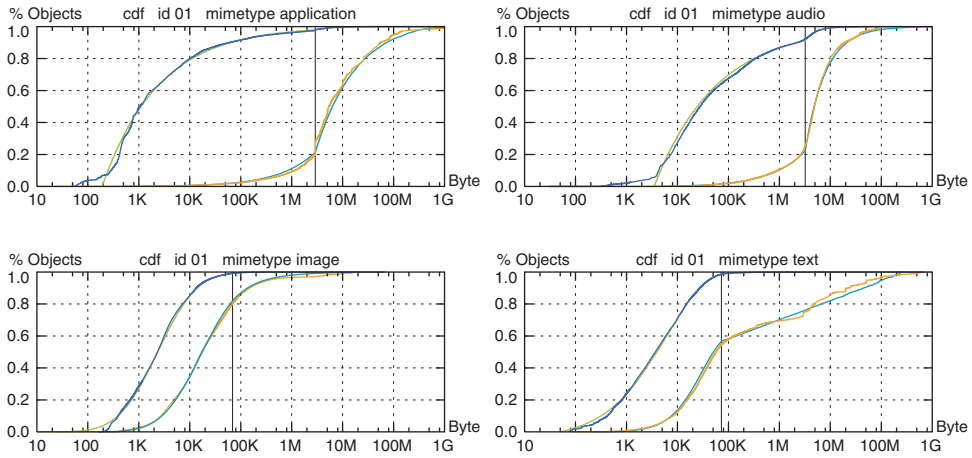
Tabelle 4.3 enthält eine Übersicht der approximierten Verteilungsfunktionen für HTTP-Responses mit den Statuscode 200. Dargestellt sind neben den gewählten Verteilungsfunktionen für Body und Tail die ermittelten Funktionsparameter sowie die Werte der Gütekriterien. Die aufgeführten Verteilungsfunktionen werden in dem

Lastgenerator durch die Funktion `set_Objectsize_200 ( id, mimetype )` abgebildet.

Mimetype		**Verteilung**	T	$\hat{F}_{Req}(T)$	$\hat{F}_{Vol}(T)$	D_{KS}	D_A
Application	Body	Pareto	2.907.935	97,59%	20,83%	0,020	0,003
	Tail	Pareto					
Audio	Body	Pareto	3.216.056	92,16%	25,40%	0,169	0,014
	Tail	Pareto					
Image	Body	Lognormal	69.100	99,15%	80,23%	0,025	0,001
	Tail	Pareto					
Text	Body	Weibull	82.142	98,95%	56,02%	0,015	0,001
	Tail	Pareto					
Video	Body	Weibull	10.038.097	97,49%	62,17%	0,100	0,032
	Tail	Pareto					
Other	Body	Lognormal	126.457	91,53%	4,85%	0,061	0,022
	Tail	Weibull					

Tabelle 4.3: Approximation der Größenverteilung von Nutzdaten

Die gute Qualität der Approximationen nach Tabelle 4.3 wird mit den Diagrammen in Abbildung 4.11 belegt. Abgesehen von schwer modellierbaren Sprüngen, die besonders bei den Mimetypes Video und Other zu erkennen sind, zeigen sich gute Approximationen auch für die übertragenen Datenvolumen.



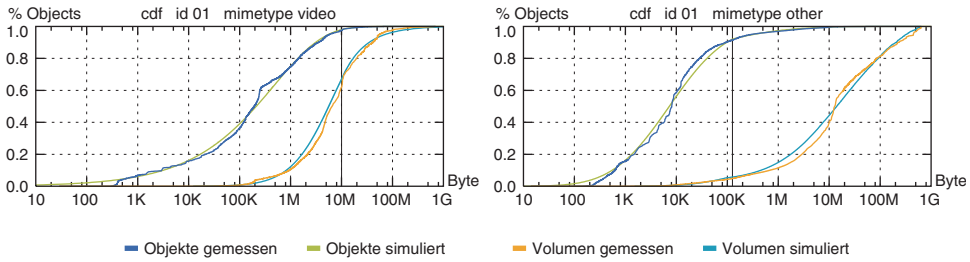


Abbildung 4.11: Verteilungen der Objektgrößen von Nutzdaten

4.2.8.2 Größenverteilung der Kontrollnachrichten

Die Größenverteilungen der Kontrollnachrichten können im Gegensatz zu den Nutzdaten über eine einzige Verteilungsfunktion für jeden Statuscode approximiert werden, da der jeweilige Ereignisraum einen wesentlich geringeren Wertebereich umfasst. Diese Erwartung wird durch die in Tabelle 4.4 aufgeführten Ergebnisse bestätigt, nach denen die Kontrollnachrichten überwiegend durch Normalverteilungen approximiert werden können.

Statuscode	**Funktion**	M	D_{KS}	D_A
302	Normal	1.161	0,049	0,013
304	Normal	516	0,221	0,044
404	Pareto	88.639	0,110	0,034
500	Normal	3.007	0,401	0,092
503	Normal	2.213	0,097	0,034
504	Normal	2.258	0,172	0,046
999	Pareto	46.469	0,083	0,037

Tabelle 4.4: Approximation der Größenverteilung von Kontrollnachrichten

Die Verwendung der Pareto-Verteilung für die Statuscodes 404 und 999 ist nicht auf deren long-tail Eigenschaft zurückzuführen, sondern lediglich auf den geeigneten Verlauf innerhalb des Ereignisraumes. Bei der Implementierung ist zu berücksichtigen, dass Funktionen mit long-tail Eigenschaften keine Werte deutlich außerhalb des tatsächlichen Ereignisraumes liefern dürfen. Daher wird in Tabelle 4.4 eine maximale obere Grenze M für jeden Statuscode angegeben. Generierte Kontrollnachrichten mit Größen oberhalb dieser Grenze werden in dem Lastgenerator verworfen. Die grafische Darstellung der Größenverteilungen in Abbildung 4.12 belegt die unterschiedlichen Datenvolumen der Kontrollnachrichten.

Die zum Teil ausgeprägten Sprünge in den Verteilungen sind auf die häufige Verwendung standardisierter Fehlertexte in WWW-Seiten zurückzuführen. Diese Aus-

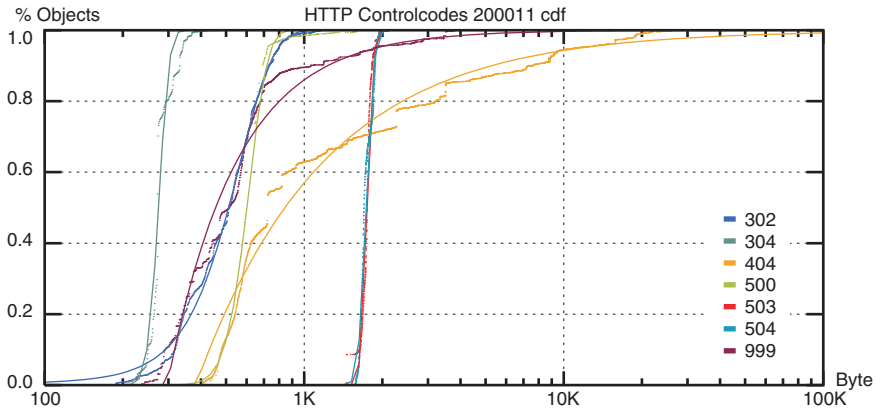


Abbildung 4.12: Verteilungen der Objektgrößen von Kontrollnachrichten

prägungen werden besonders an denjenigen Kontrollnachrichten deutlich, die sich geeignet durch Normalverteilungen approximieren lassen. Da sich die zum Teil hohen Werte von D_{KS} ebenfalls durch die Sprünge erklären lassen, deren genaue Approximation für die Modellierung unerheblich ist, wird hier als ausschlaggebendes Gütekriterium ausschließlich D_A betrachtet.

4.2.9 Mittlere Hauptverkehrsstunde

In Analogie zu den Belegungen in herkömmlichen Telekommunikationsnetzen ergibt sich die mittlere Hauptverkehrsstunde im DFN-Cache-Verbund aus der Betrachtung der Übertragungsdauern bzw. der gleichzeitig bearbeiteten HTTP-Responses [ITG97]. Um die Bedeutung und Aussagekraft der mittleren Hauptverkehrsstunde für die hier vorliegende Problemstellung zu prüfen, wird im Folgenden auch die Anzahl der im Mittel eingehenden HTTP-Requests (Belegungswünsche) sowie die mittlere Summendatenrate für den Verkehrsfluss Out_{WWW} betrachtet.

Da auf den DFN-Cache-Servern an Wochenenden ein deutlich geringeres Verkehrsaufkommen zu beobachten war, erfolgt die Berechnung der mittleren Hauptverkehrsstunde ausschließlich für die Wochentage Montag bis Freitag. Daraus folgt, dass aus jedem Datensatz 20 Tage in die Berechnungen einfließen. Für die betrachteten Werte werden jeweils Minutenmittel aggregiert, so dass die Angabe der entsprechenden Hauptverkehrsstunden mit einer Auflösung von einer Minute möglich ist. Eine Übersicht der berechneten mittleren Hauptverkehrsstunden aller Datensätze zeigt Tabelle 4.5.

	ID	eingehende HTTP-Requests	bearbeitete HTTP-Responses	Datenrate
Hauptverkehrsstunde	97	12:51–13:50	14:14–15:13	12:27–13:26
	98	12:59–13:58	14:34–15:33	14:42–15:41
	99	13:24–14:23	14:03–15:02	12:40–13:39
	00	12:28–13:27	13:39–14:38	10:43–11:42
	01	12:37–13:36	13:32–14:31	13:05–14:04
Stunde geringsten Verkehrs	97	05:30–6:29	05:35–06:34	05:34–06:33
	98	05:06–6:05	05:31–06:30	05:25–06:24
	99	05:20–6:19	05:31–06:30	05:28–06:27
	00	05:01–6:00	05:08–06:07	05:13–06:12
	01	05:01–6:00	05:13–06:12	05:14–06:13

Tabelle 4.5: Mittlere Hauptverkehrsstunden

Die Verläufe der für den Datensatz 01 betrachteten Tagesmittel sind in Abbildung 4.13 dargestellt. Neben den Hauptverkehrsstunden sind auch die Stunden mit dem geringsten Verkehrsaufkommen grau hinterlegt.

Zunächst zeigen die Diagramme das zu erwartende Tagesprofil. Im Gegensatz z. B. zu ISDN-Netzen kommerzieller Anbieter mit mehreren tarifbedingten Ausprägungen liegen die Hauptverkehrsstunden im Wissenschaftsnetz eindeutig zwischen 12:00 bis 15:00 Uhr, der Verkehr im Zeitraum von ca. 10:00 bis 18:00 Uhr verläuft durchgehend oberhalb des Tagesmittels. Ein Vergleich der Diagramme in Abbildung 4.13 sowie der Angaben in Tabelle 4.5 zeigt, dass in allen betrachteten Datensätzen das höchste Aufkommen eingehender HTTP-Requests nicht mit der mittleren Hauptverkehrsstunde zusammentrifft, sondern unmittelbar davor festgestellt werden kann. Die Ursachen für hohe Übertragungsdauern, die zu einer Verschiebung der mittleren Hauptverkehrsstunde führen können, sind jedoch im Wesentlichen auf den WWW-Servern außerhalb des Wissenschaftsnetzes und auf den entsprechenden Netzsegmenten zu finden. Die Bewertung einer zeitlichen Korrelation zwischen den eingehenden HTTP-Requests und den Übertragungs- bzw. Bearbeitungsdauern ist deshalb ohne tiefergehende Untersuchungen nicht möglich.

4.2.10 Datenraten

Aus der Betrachtung der mittleren Hauptverkehrsstunden lassen sich die Datenraten ermitteln, mit denen die Objekte von den DFN-Cache-Servern an die lokalen Cache-Server übertragen wurden. Nach den Vorüberlegungen aus Kapitel 4.2.2.2 sind da-

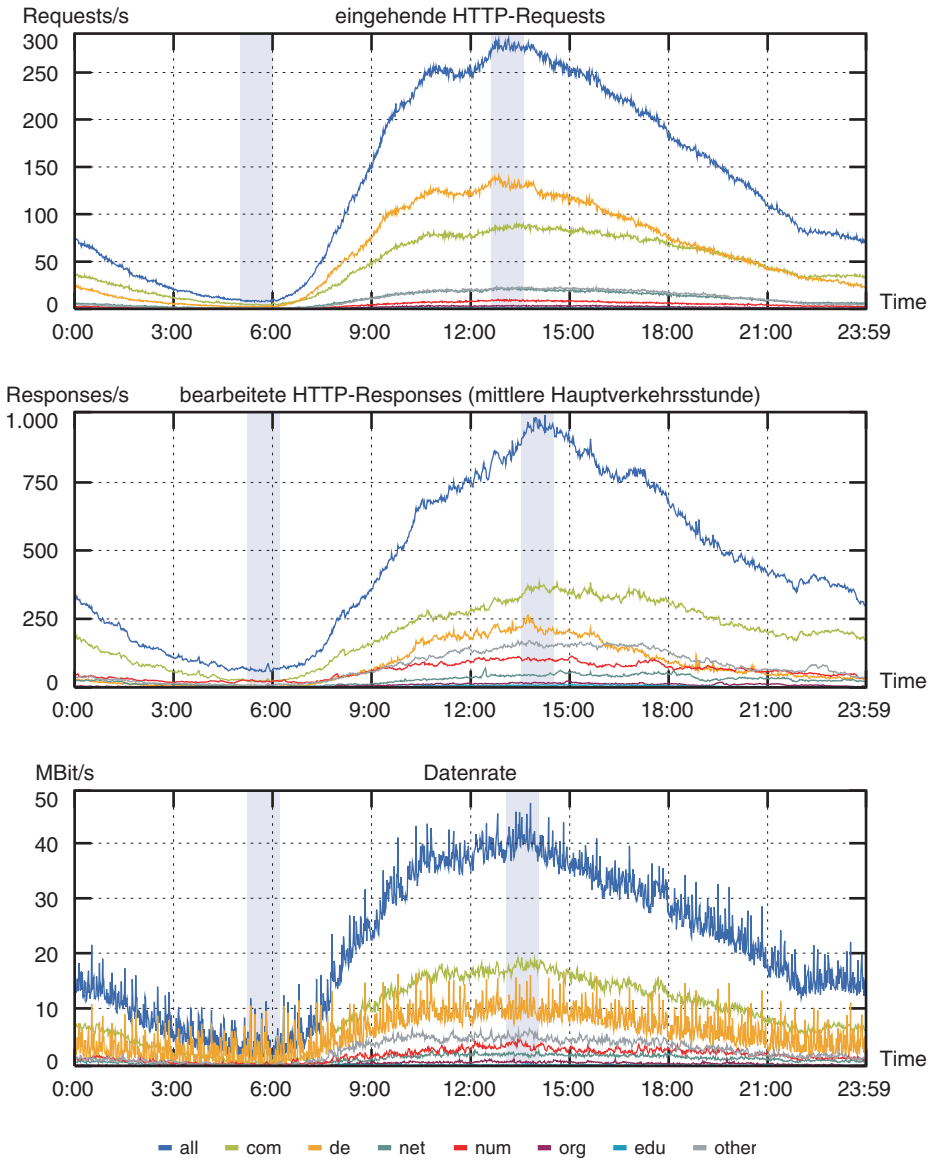


Abbildung 4.13: Mittlere Hauptverkehrsstunde

bei sowohl Toplevel Domains, Statuscodes als auch die in Kapitel 3.2.4 betrachteten Arten von Cache-Hits und Cache-Misses zu berücksichtigen. Die Bestimmung der Datenströme, die für den Datensatz 01 zu modellieren sind, wurde bereits bei der Erstellung von Tabelle 4.2 erläutert. Um die Darstellung der untersuchten Datenraten übersichtlich zu halten, werden in Abbildung 4.14 zunächst die übertragenen Nutzdaten, d. h HTTP-Responses mit den Statuscodes 200 und 304, betrachtet.

Aus den Diagrammen lassen sich unmittelbar mehrere quantitative Aussagen ableiten:

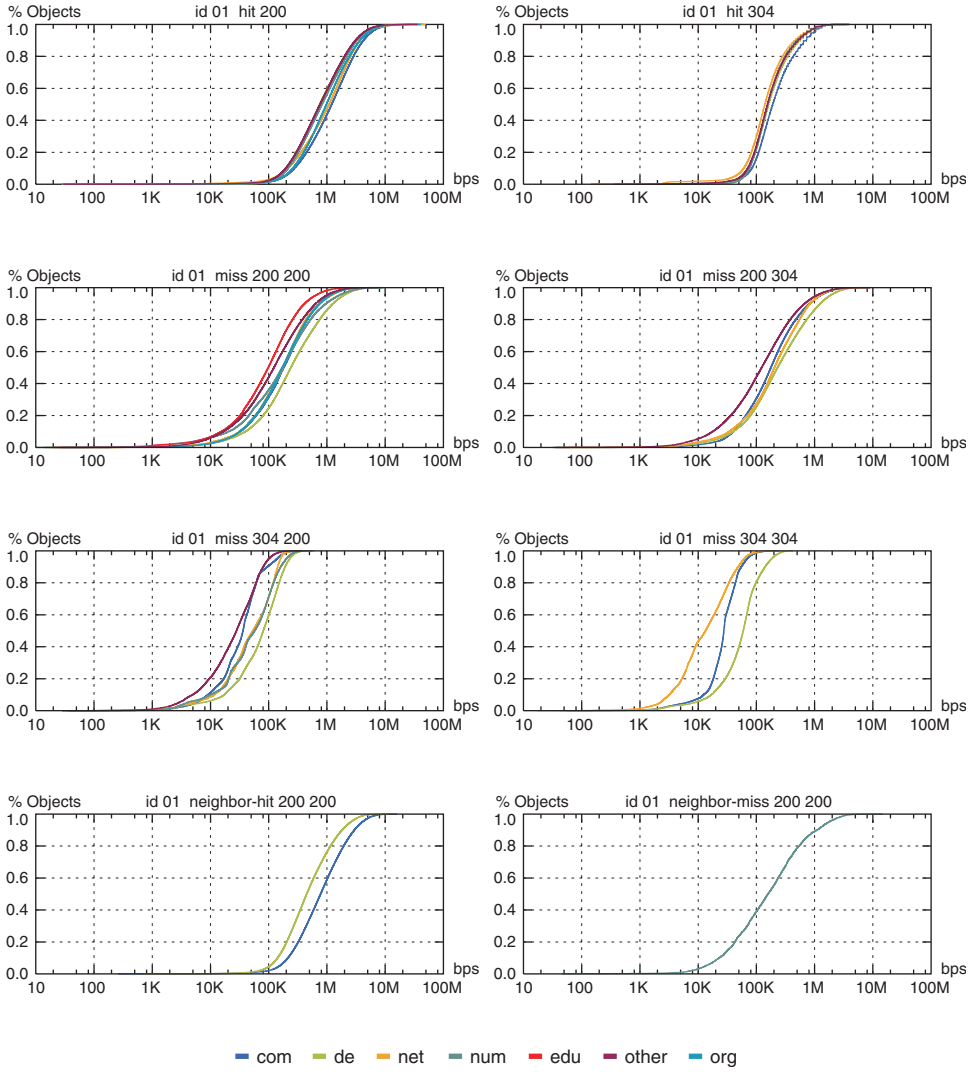


Abbildung 4.14: Datenraten HTTP-Responses mit Statuscodes 200 und 304

- Die Übertragungsrate von Cache-Hits (hit 200) liegt ca. eine Größenordnung über der von Cache-Misses (miss 200 200).
- Wie in den Vorüberlegungen bereits vermutet wurde, ist die Übertragungsrate von Cache-Hits weitgehend unabhängig von der jeweiligen Toplevel Domain.
- Im Gegensatz dazu weisen die Datenraten von Cache-Misses deutliche Unterschiede zwischen verschiedenen Toplevel Domains auf.
- Cache-Hits auf benachbarten Cache-Servern innerhalb des DFN-Cache-Verbundes (neighbor-hit 200 200) werden nur mit unwesentlich geringerer Datenrate übertragen als direkte Cache-Hits.
- Demgegenüber werden bestätigte Cache-Hits (miss 200 304) lediglich mit annähernd gleicher Übertragungsrate übertragen wie Cache-Misses.

Die vergleichbar niedrigen Datenraten von HTTP-Responses mit Statuscodes 304 (hit 304 und miss 304 304) sind durch die geringen Datenvolumen zu erklären, die mit diesen HTTP-Responses übertragen werden (Abbildung 4.12). Mit einem mittleren Volumen von 516 Byte sind die resultierenden Übertragungsdauern dieser HTTP-Responses jedoch wesentlich niedriger gegenüber der Übertragung vollständiger Objekte. Neben der Schonung von Ressourcen wird hierdurch ein weiterer Vorteil bedingter HTTP-Requests bestätigt.

Die Ergebnisse der Approximation sind in Tabelle 4.6 aufgeführt. Der Tabelle ist zu entnehmen, dass sämtliche Datenraten durch Lognormal- oder Weibull-Verteilungen geeignet approximiert werden können. Die Gütekriterien weisen darauf hin, dass die Datenraten der übertragenen Nutzdaten mit diesen Verteilungen durchgehend in guter Näherung modelliert werden können.

	Statuscode				Toplevel Domain	Verteilung	D_{KS}	D_A
	C_L-C_1	C_1-S	C_1-C_2	C_2-S				
Hit	200	–	–	–	com	lognormal	0,027	0,011
					de	lognormal	0,023	0,011
					net	weibull	0,033	0,008
					num	lognormal	0,019	0,010
					org	lognormal	0,010	0,005
					other	lognormal	0,014	0,007
	304	–	–	–	com	lognormal	0,052	0,023
					de	lognormal	0,053	0,023
					net	lognormal	0,046	0,022
					num	lognormal	0,059	0,023
					other	lognormal	0,055	0,020
Miss	200	200	–	–	com	weibull	0,024	0,014
					de	lognormal	0,017	0,008
					net	lognormal	0,027	0,013
					num	weibull	0,025	0,014
					edu	weibull	0,021	0,013
					org	lognormal	0,018	0,008
					other	lognormal	0,026	0,009
	200	304	–	–	com	lognormal	0,011	0,004
					de	lognormal	0,018	0,006
					net	weibull	0,022	0,012
					other	lognormal	0,019	0,007
	304	200	–	–	com	weibull	0,055	0,013
					de	weibull	0,020	0,011
					net	weibull	0,048	0,017
					num	weibull	0,029	0,014
					other	weibull	0,013	0,007
	304	304	–	–	com	lognormal	0,055	0,017
					de	weibull	0,037	0,019
					other	lognormal	0,042	0,015

Tabelle 4.6: Approximation der Datenraten für Statuscodes 200 und 304

Neighbor-Hit	200	–	200	–	com	lognormal	0,012	0,006
					de	lognormal	0,027	0,013
Neighbor-Miss	200	–	200	200	num	lognormal	0,014	0,006

Tabelle 4.6: Approximation der Datenraten für Statuscodes 200 und 304

Nach Tabelle 4.2 werden für den Datensatz 01 die Datenraten von Kontrollnachrichten lediglich für Cache-Misses betrachtet. Die entsprechenden Diagramme in Abbildung 4.15 zeigen deutliche Differenzen zwischen den einzelnen Datenströmen, sowohl allgemein zwischen den verschiedenen Statuscodes als auch zwischen unterschiedlichen Toplevel Domains bei Kontrollnachrichten mit identischen Statuscodes.

Bereits in Kapitel 4.2.2.2 wurde die Modellierung von Kontrollnachrichten mit dem Statuscode 504 durch die Angabe eines fest vorgegebenen Timeouts erörtert. Die hier dargestellte Übertragungsrate von ca. 60 Bit/s ergibt mit dem mittleren Volumen von 2.258 Byte aus Tabelle 4.4 eine mittlere Übertragungsdauer von 301 Sekunden bzw. 5 Minuten. Weiterhin weisen die geringen und ungleichmäßig verlaufenden Datenraten der Kontrollnachrichten mit den Statuscodes 500, 503 und 999 auf schwankende und zum Teil signifikante Verzögerungen auf Netzsegmenten oder WWW-Servern hin.

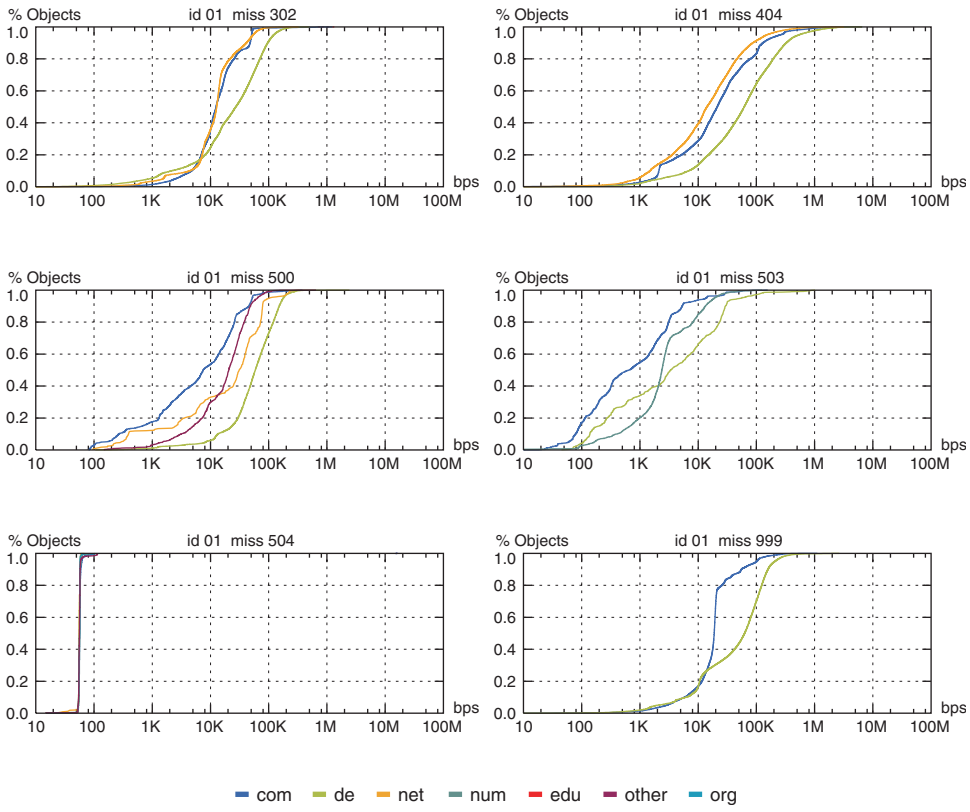


Abbildung 4.15: Datenraten Kontrollnachrichten

Die Approximation der Datenraten für Kontrollnachrichten ergibt sich analog zu der Betrachtung der übertragenen Nutzdaten. Die Datenraten lassen sich auch hier geeignet durch Lognormal- und Weibull-Verteilungen nachbilden. Aufgrund der ungleichmäßigen Verläufe ist die Qualität der Approximationen jedoch eingeschränkt, was an den höheren Werten der Gütekriterien abzulesen ist.

	Statuscode				Toplevel Domain	Verteilung	D_{KS}	D_A
	C_L-C_1	C_1-S	C_1-C_2	C_2-S				
Miss	302	302	–	–	com	lognormal	0,079	0,019
					de	weibull	0,027	0,015
					net	lognormal	0,102	0,033
	404	404	–	–	com	lognormal	0,069	0,015
					de	weibull	0,026	0,009
					other	weibull	0,037	0,011
	500	500	–	–	com	weibull	0,046	0,023
					de	weibull	0,027	0,012
					num	weibull	0,128	0,043
					other	weibull	0,051	0,022
	503	503	–	–	com	lognormal	0,049	0,013
					de	weibull	0,052	0,011
					num	lognormal	0,097	0,041
	504	504	–	–	com	Timeout 300 Sekunden		
					de			
					net			
					num			
					org			
					other			
	999	999	–	–	com	weibull	0,079	0,032
					de	lognormal	0,079	0,019

Tabelle 4.7: Approximation der Datenraten für Kontrollnachrichten

Obwohl Cache-Hits von Kontrollnachrichten bei der Modellierung des Datensatzes 01 nicht betrachtet werden, soll ein Vergleich auf den potentiellen Nutzen des Caching von Kontrollnachrichten hinweisen. In Abbildung 4.16 werden die Datenraten von Cache-Hits sämtlicher Kontrollnachrichten dargestellt, die in der betrachteten Hauptverkehrsstunde übertragen wurden. Im Vergleich mit Abbildung 4.15 zeigen sich um bis zu zwei Größenordnungen höhere Übertragungsraten, was einer erheblichen Reduzierung von Wartezeiten gleichzusetzen ist.

Abschließend zeigt ein Vergleich zwischen der mittleren Hauptverkehrsstunde und der Stunde des geringsten Verkehrsaufkommens, dass bei der Betrachtung der Datenraten unbedingt die gegenwärtige Lastsituation zu berücksichtigen ist. In Abbildung 4.17 werden die Datenraten von Cache-Hits und Cache-Misses aus der Toplevel Domain `de` für die Datensätze 00 und 01 gegenübergestellt. Es zeigt sich, dass für den Datensatz 00 Cache-Hits in der Hauptverkehrsstunde mit deutlich geringerer Datenrate übertragen werden als während der Stunde geringsten Verkehrsaufkommens. Demgegenüber zeigt ein Vergleich der Datenraten von Cache-Hits

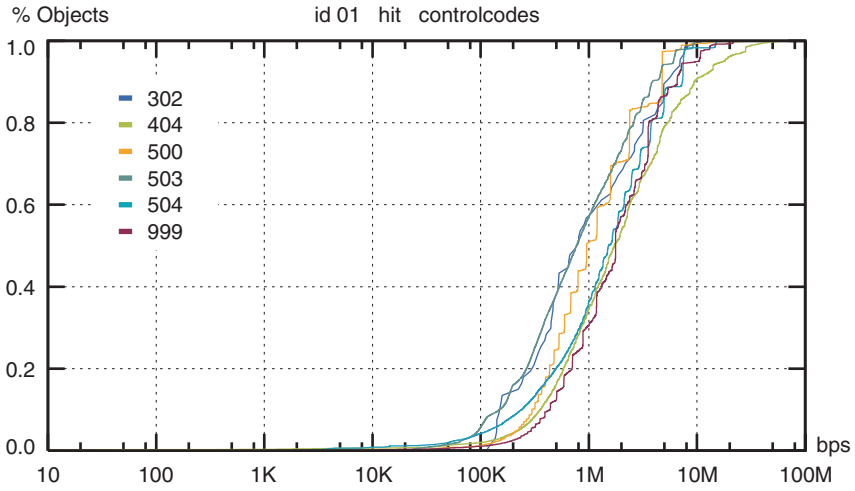


Abbildung 4.16: Datenraten Cache-Hits von Kontrollnachrichten

und Cache-Misses in der Stunde des geringsten Verkehrsaufkommens zwischen beiden betrachteten Datensätzen einen annähernd identischen Verlauf. Daraus folgt, dass die alten DFN-Cache-Server bei geringer Last Objekte mit hohen Datenraten übertragen konnten, mit steigender Last jedoch eine deutliche Verringerung der Übertragungsqualität einherging. Die Verteilungen für den Datensatz 01 zeigen den allgemein zu erwartenden Verlauf für nicht überlastete Cache-Server. Cache-Hits werden in beiden betrachteten Stunden mit höheren Datenraten ausgeliefert als Cache-Misses.

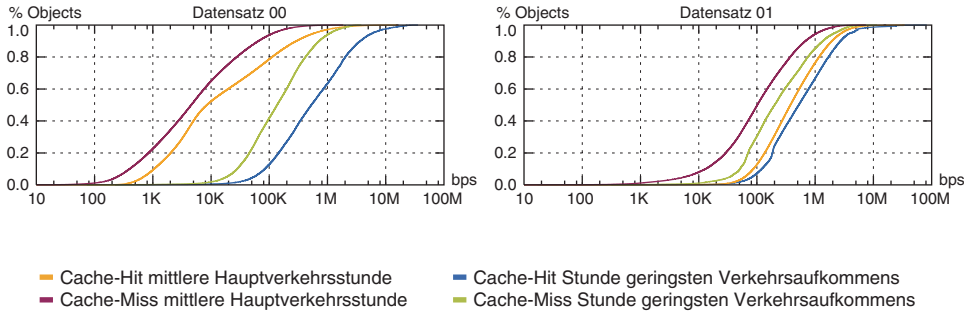


Abbildung 4.17: Datenraten aus Toplevel Domain `de` für Datensätze 00 und 01

4.2.11 Zwischenankunftszeiten

Um die Zwischenankunftszeiten innerhalb des Zeitfensters für den gesamten Verkehrsfluss Out_{WWW} angeben zu können, müssten die Einträge der Logdateien aller DFN-Cache-Server in eine einheitliche zeitliche Ordnung gebracht werden. Diese Ordnung lässt sich jedoch aufgrund der fehlenden Synchronität der Rechneruhren nicht erreichen. Die Uhren der DFN-Cache-Server waren zwar über NTP-Timeserver synchronisiert, eine gegenseitige variierende Abweichung im Bereich von

10 – 50 ms konnte jedoch nicht verhindert werden. Da ein erheblicher Anteil der auf einzelnen DFN-Cache-Servern gemessenen Zwischenankunftszeiten bereits deutlich unterhalb von 10 ms lag, kann hier zunächst keine vollständige Betrachtung der Zwischenankunftszeiten für den gesamten Verkehrsfluss Out_{WWW} durchgeführt werden. Alternativ bietet sich zuerst die Betrachtung einzelner Cache-Server an.

Tabelle 4.8 enthält die Anzahl der eingehenden HTTP-Requests, die auf den einzelnen DFN-Cache-Servern während der mittleren Hauptverkehrsstunde gezählt wurden. Ein Vergleich der Werte zeigt zunächst ein deutlich geringeres Aufkommen für die DFN-Cache-Server cs-han10 und cs-han21, die die Partition für die logische Toplevel Domain `abroad` bildeten. Nach Kapitel 3.1.2.3 wurde die Partitionierung im neuen DFN-Cache-Verbund anhand des übertragenen Datenvolumens und nicht anhand der bearbeiteten HTTP-Requests vorgenommen. Die Zuordnung reeller Toplevel Domains wurde dabei so gewählt, dass in jeder Partition mit annähernd gleichem Datenvolumen zu rechnen war. Die in Tabelle 4.8 aufgeführten HTTP-Requests belegen daher erneut, dass das übertragene Datenvolumen keine Rückschlüsse auf die Anzahl der bearbeiteten HTTP-Requests zulässt. Darüber hinaus ist auch zwischen den DFN-Cache-Servern innerhalb einer Partition eine deutliche Differenz zu erkennen. Die Ursache hierfür liegt offenbar in einer ungleichmäßigen Zuordnung der lokalen Cache-Server.

DFN-Cache-Server	HTTP-Responses in HvStd über 20 Tage	HTTP-Requests / ms
cs-han10	3.927.455	0,05455
cs-han11	6.329.620	0,08792
cs-han20	7.668.516	0,10651
cs-han21	3.404.734	0,04729

Tabelle 4.8: Bearbeitete HTTP-Responses in mittlerer Hauptverkehrsstunde

Ein Hinweis auf die erforderliche Genauigkeit, mit der die Zwischenankunftszeiten bestimmt werden müssen, ergibt sich aus der Betrachtung der Übertragungsdauer d. Die mittlere Übertragungsdauer $\bar{d}$ entspricht der mittleren Bedienzeit $1/\mu$. In Abbildung 4.18 wird die Verteilung der Übertragungsdauer für den gesamten Zeitraum und für die mittlere Hauptverkehrsstunde betrachtet. Es zeigt sich, dass in der mittleren Hauptverkehrsstunde ca. 90% der Objekte mit einer Übertragungsdauer von mehr als 50 ms übertragen werden. Die mittlere Übertragungsdauer beträgt $\bar{d}_{\mathrm{HvStd}} = 3{,}2691$ s bzw. $\bar{d}_{\mathrm{Gesamt}} = 2{,}0417$ s. Für die Hauptverkehrsstunde ergibt sich daraus eine mittlere Bedienrate von $\mu_{\mathrm{HvStd}} = 0{,}3058$ Requests/s.

Die hinterlegten approximierten Lognormalverteilungen sowie die Lage der Mittelwerte weisen darauf hin, dass auch die Verteilung der Übertragungszeiten long-tail Eigenschaften aufweist. Diese Feststellung wird bei der Betrachtung der Stabilität des Simulationsmodells in Kapitel 5.3.3 aufgegriffen.

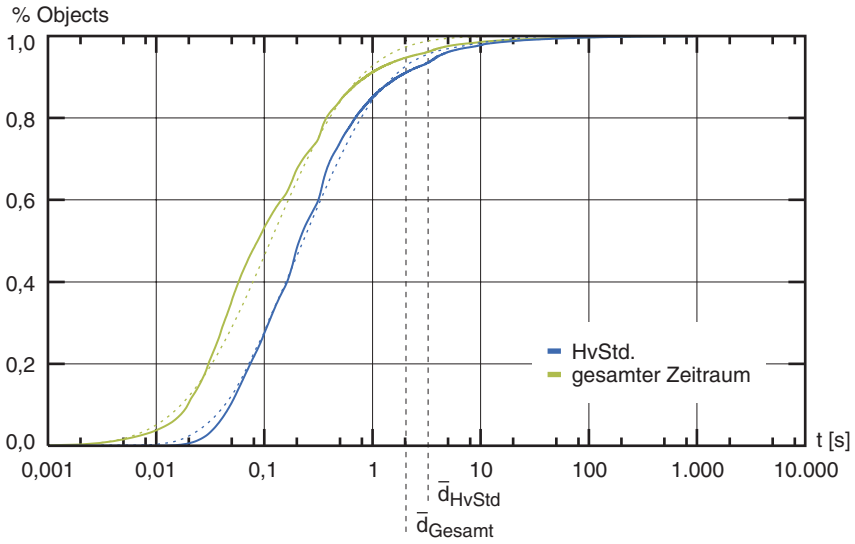


Abbildung 4.18: Verteilung der Übertragungsdauer

Der Verlauf der Zwischenankunftszeiten für die einzelnen DFN-Cache-Server wird mit den entsprechenden Approximationen in Abbildung 4.19 dargestellt. Bei der Auswertung war zu berücksichtigen, dass die Zeitinformationen in den Logdateien mit einer Auflösung von einer Millisekunde angegeben sind. HTTP-Requests, die innerhalb derselben Millisekunde von einem DFN-Cache-Server empfangen wurden, ergaben folglich eine Zwischenankunftszeit von 0 ms. Um diese Werte auch in der Darstellung geeignet zu berücksichtigen, werden sie willkürlich zum Zeitpunkt $T = 0{,}1\,\text{ms}$ eingetragen.

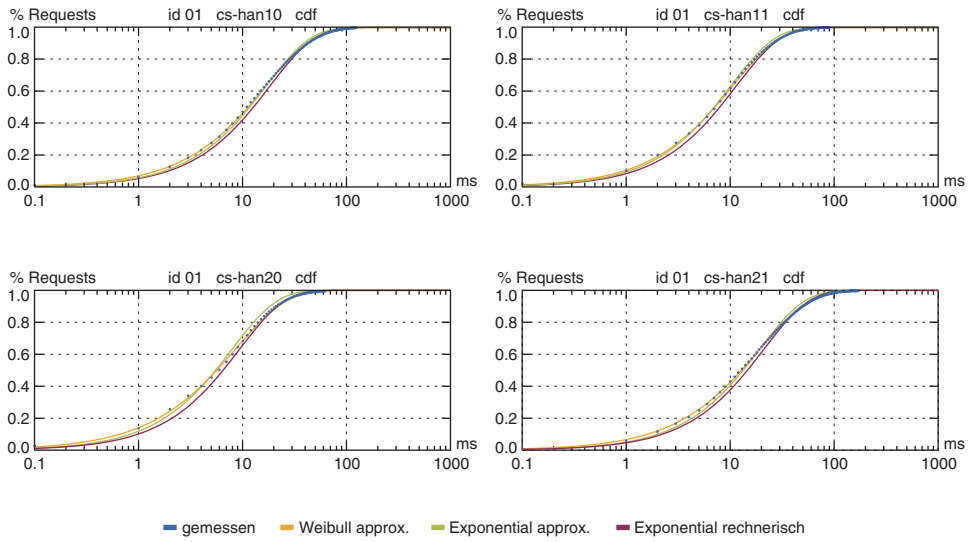


Abbildung 4.19: Verteilung der Zwischenankunftszeiten

Bereits aus den Diagrammen ist zu erkennen, dass sowohl die Weibull-Verteilung als auch die Exponentialverteilung geeignete Approximationen liefern. Die Verteilung der Zwischenankunftszeiten innerhalb des gesamten Verkehrsflusses Out_{WWW} lässt sich aus den gewonnenen Ergebnissen jedoch nicht unmittelbar ableiten.

Es ist bekannt, dass die mittleren Zwischenankunftszeiten verschiedener Ankunftsprozesse überlagert werden können, wenn die einzelnen Ankünfte unabhängig voneinander sind. Diese Bedingung wird ausschließlich von Poisson-Prozessen erfüllt [TaKa98]. Da die Zwischenankunftszeiten von Poisson-Prozessen exponentialverteilt sind und der Parameter λ der mittleren Ankunftsrate entspricht, können mit den Werten aus Tabelle 4.8 die entsprechenden Exponentialverteilungen in die Diagramme übernommen werden. Ein Vergleich der Gütekriterien in Tabelle 4.9 zeigt, dass die so ermittelten Abbildungen eine geringere Qualität liefern als die berechneten Approximationen.

DFN-Cache-Server	Approximation				Messung	
	Weibull		Exponential		Exponential	
	D_{KS}	D_A	D_{KS}	D_A	D_{KS}	D_A
cs-han10	0,01001	0,00033	0,01001	0,00054	0,05052	0,01913
cs-han11	0,00995	0,00038	0,02097	0,00082	0,04553	0,01988
cs-han20	0,01460	0,00078	0,03510	0,00180	0,06892	0,02338
cs-han21	0,00760	0,00021	0,02562	0,00060	0,05423	0,01909

Tabelle 4.9: Approximation der Zwischenankunftszeiten

Wird nach den vorangegangenen Betrachtungen der Ankunftsprozess aller von lokalen Cache-Servern eingehenden HTTP-Requests als Poisson-Prozess angesehen, ergibt sich mit den Werten aus Tabelle 4.8 die gesamte Ankunftsrate für den Datensatz 01 wie folgt:

$$\lambda_{\mathrm{HvStd}} = \lambda_{\mathrm{cs\text{-}han10}} + \lambda_{\mathrm{cs\text{-}han11}} + \lambda_{\mathrm{cs\text{-}han20}} + \lambda_{\mathrm{cs\text{-}han21}} = 0{,}2963 \text{ Requests/ms} \qquad (4.7)$$

Daraus folgt die mittlere Zwischenankunftszeit unmittelbar zu $1/\lambda_{\mathrm{HvStd}} = 3{,}3756$ ms. Eine entsprechende Rechnung für die approximierten Exponentialverteilungen liefert eine um 8% höhere mittlere Ankunftsrate von $\lambda_{\mathrm{approx}} = 0{,}3207$ Requests/ms.

Da weitere Informationen über die Verteilung der Zwischenankunftszeiten nicht vorliegen und die mittleren Zwischenankunftszeiten wesentlich geringer als der überwiegende Teil der in Abbildung 4.18 dargestellten Übertragungsdauern sind, werden für den weiteren Verlauf der Arbeit exponentialverteilte Zwischenankunftszeiten angenommen.

Aus dem Satz von Little [Lit61] ergibt sich mit der mittleren Bedienrate von $\mu_{HvStd} = 0{,}3058$ Requests/s die Abschätzung der im Mittel belegten Bedieneinheiten zu $\overline{N}_{HvStd} = \lambda_{HvStd} / \mu_{HvStd} = 969$. Die entsprechende Anzahl der in der mittleren Hauptverkehrsstunde gleichzeitig bearbeiteteten HTTP-Responses wird ebenso durch die in Abbildung 4.13 dargestellten Tageskurven bestätigt wie die mittlere Ankunftsrate von $\lambda_{HvStd} = 296{,}3$ Requests/s.

Neben den HTTP-Requests der lokalen Cache-Server treten in einem Cache-Verbund auch HTTP-Requests benachbarter Cache-Server auf. Da das Aufkommen dieser HTTP-Requests nicht unabhängig von den HTTP-Requests der lokalen Cache-Server ist, sondern vielmehr durch die Trefferraten auf den Neighbors bestimmt wird, kann der resultierende Ankunftsprozess aller HTTP-Requests für einen einzelnen Cache-Server nicht ohne weiterführende Untersuchungen als Poisson-Prozess angesehen werden. Diese Einschränkung ist lediglich für eine analytische Betrachtung relevant, nicht jedoch für die Implementierung des Simulationsmodells.

Abschließend ist anzumerken, dass mit der Darstellung des Ankunftsprozesses als Poisson-Prozess die Feststellung in Kapitel 4.2.2.3 belegt wird, nach der Lastspitzen nicht durch lokale Cache-Server weitergeleitet werden können. Der auf den übergeordneten Cache-Servern eintreffende Verkehr ist dadurch bereits geglättet und weist lediglich eine geringe Burstiness auf. Demgegenüber wäre Verkehr mit ausgeprägter Burstiness nicht geeignet durch Poisson-Prozesse darstellbar [PaFl95].

4.2.12 Klienten

Abschließend wird untersucht, nach welcher Verteilung das Verkehrsaufkommen den verursachenden Klienten bzw. lokalen Cache-Servern zugeordnet werden kann. Da die generierte Last nicht zufällig auf die DFN-Cache-Server, sondern über die Konfiguration determiniert verteilt wird, muss das Verhalten lokaler Cache-Server unterschiedlicher Kapazität nachgebildet werden. Die Angabe der entsprechenden Verteilungen wird aus den von den DFN-Cache-Servern empfangenen HTTP-Requests ermittelt. Um auszuschließen, dass einzelne Nutzer oder singuläre Tests die Verteilungen verfälschen, werden nur die lokalen Cache-Server betrachtet, von denen an mindestens 27 Tagen jeweils 500 HTTP-Requests empfangen wurden. Die Anzahl der so ermittelten, verschiedenen lokalen Cache-Server wurde bereits in Tabelle 3.11 angegeben, die entsprechende Darstellung der Verteilungen für alle Datensätze enthält Abbildung 4.20.

Die Approximation erfolgt in guter Näherung für die Bereiche unterhalb 95% mit der Weibull-Verteilung. Da auch hier vermieden werden muss, dass durch die long-

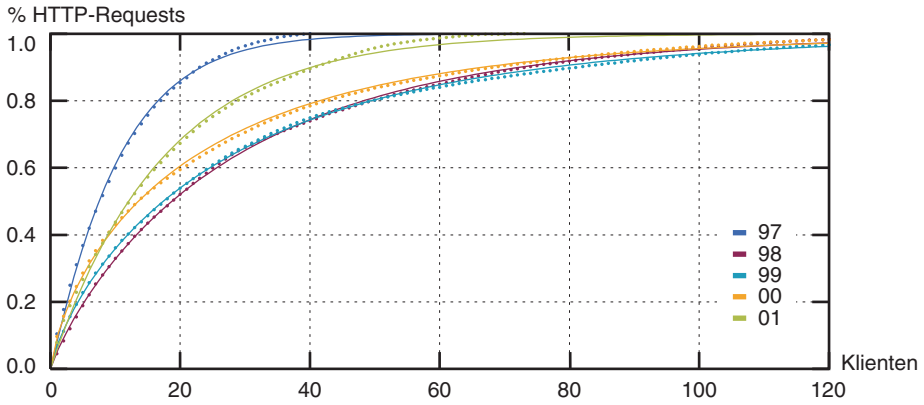


Abbildung 4.20: Verteilung des Verkehrsaufkommens nach Klienten

tail Eigenschaft der Weibull-Verteilung Werte außerhalb des Ereignisraumes generiert werden, wird ein Maximums M angegeben.

ID	Funktion	M	D_{KS}	D_A
97	Weibull	40	0,02712	0,01173
98	Weibull	151	0,02532	0,00834
99	Weibull	180	0,03172	0,00754
00	Weibull	165	0,01733	0,00708
01	Weibull	73	0,02755	0,01141

Tabelle 4.10: Approximation der Größenverteilung von Kontrollnachrichten

Die Verteilung der Klienten wird in dem Lastgenerator durch die Funktion `set_client ( id )` abgebildet.

Kapitel 5

Implementierung

Der zuverlässige Betrieb des DFN-Cache-Verbundes erforderte ein kontinuierliches Monitoring, um die aktuelle Lastsituation auf allen DFN-Cache-Servern geeignet beurteilen zu können. Eine detailliertere Betrachtung der Last auf den DFN-Cache-Servern und des Verkehrsaufkommens auf den Netzsegmenten ergab sich aus den nachträglichen Auswertungen der Logdateien. Eine zuverlässige Bewertung von Konfigurationsänderungen an dem DFN-Cache-Verbund *vor* der Umsetzung war jedoch nicht möglich, da hierfür die entsprechenden Werkzeuge zur Berechnung, Modellierung und Simulation fehlten. In diesem Kapitel wird ein Simulationsmodell entworfen und implementiert, um derartige Untersuchungen durchführen zu können. Mit Bezug auf den DFN-Cache-Verbund sollen folgende Aspekte anhand des Modells untersucht werden:

- Verteilung der Last auf die DFN-Cache-Server,
- Verteilung des Verkehrsaufkommens auf den Netzsegmenten zwischen den beteiligten Cache-Servern,
- Verhalten des gesamten DFN-Cache-Verbundes bei Störungen oder Ausfällen einzelner DFN-Cache-Server.

5.1 Analyse

5.1.1 Einordnung der Problemstellung

Das Vorgehen bei dem Entwurf eines Simulationsmodells leitet sich generell von der Methode ab, mit der die Leistungsbewertung des betrachteten Systems durchgeführt werden soll. Abbildung 5.1 zeigt eine Klassifizierung von verschiedenen Methoden zur Leistungsbewertung nach [Tra96].

Eine analytische Betrachtung des Verkehrsmodells stellt die Methode mit der höchsten Effektivität und Variabilität dar, da aus den mathematischen Beziehungen direkt die betrachteten Leistungs- oder Verkehrswerte bei vorgegebenen Randbedingungen berechnet werden können [Kob78]. Der Anwendungsbereich der analytischen

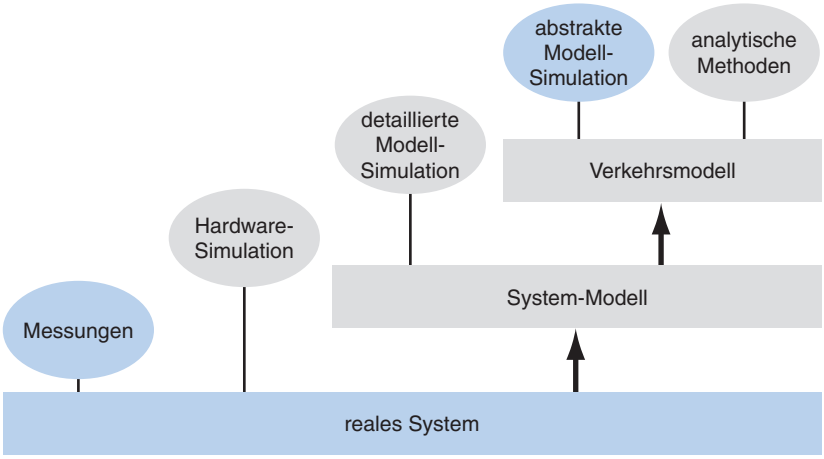


Abbildung 5.1: Methoden zur Leistungsbewertung [Tra96]

Modellierung ist jedoch begrenzt. Zum einen führen komplexe Verkehrsmodelle zu Gleichungssystemen, die nur unter starken Vereinfachungen lösbar sind. Zum anderen können nicht beliebige Verteilungsfunktionen herangezogen werden, da häufig für die Berechnung die Existenz der Laplace-Transformierten vorausgesetzt wird [Kle75]. Ist eine analytische Betrachtung nicht anwendbar, muss auf eine abstrakte oder detailgetreue Simulation des Systems zurückgegriffen werden. Die Wahl der geeigneten Methode zur Leistungsbewertung ergibt sich daher aus der Komplexität des entworfenen Modells sowie aus den Funktionen, mit denen die Eingangslasten und die Verkehrsbeziehungen mathematisch dargestellt werden.

Im Rahmen dieser Arbeit wird kein analytisches Modell für einen Cache-Verbund entworfen. Um die Auswirkungen temporärer Ausfälle geeignet untersuchen zu können, sollen während eines Simulationsdurchlaufs interaktiv einzelne Cache-Server ein- und ausgeschaltet werden. Analytische Modelle bieten weder die hierfür notwendige Interaktivität noch die Möglichkeit, die nach Störungen auftretenden transienten Vorgänge im Cache-Verbund geeignet beobachten zu können. Darüber hinaus weist ein Teil der Verteilungsfunktionen, die bei der Modellierung der Verkehrsflüsse angewendet wurden, keine geeigneten mathematischen Eigenschaften auf, die zu einer analytischen Betrachtung des Modells führen könnten (s. Kapitel 4.1.1.1).

5.1.2 Entwurf eines Warteschlangenmodells

In dem ersten Schritt zum Entwurf eines geeigneten Simulationsmodells wird das Modell eines isolierten Cache-Servers untersucht. Ziel der Betrachtung ist es, unter Berücksichtigung der vorliegenden Problemstellung und der zur Verfügung stehenden Datenquellen das Warteschlangenmodell für einen vollständigen Cache-Verbund zu bilden, das aus der Kombination von Modellen einzelner Cache-Server besteht.

5.1.2.1 Modell eines Cache-Servers

In [MeAl98] werden die an einem lokalen Cache-Server auftretenden Verkehrsflüsse durch ein geschlossenes Warteschlangenmodell beschrieben. Das in Abbildung 5.2 wiedergegebene Modell zeigt einen hohen Detaillierungsgrad. Für eine exakte Berechnung müssen Informationen über die Verteilung von jeweils fünf verschiedenen Warte- und Bedienzeiten vorliegen. Da aus den Logdateien eines Cache-Servers ein großer Teil der Informationen, die für eine Berechnung nach diesem Modell notwendig sind, nicht zur Verfügung steht, müssten verschiedene ergänzende Untersuchungen an dem System durchgeführt werden. Insgesamt würde die Verwendung dieses Modells zu einer derart komplexen Darstellung eines Cache-Verbundes führen, aus der weder handhabbare analytische Methoden noch implementierbare Simulationsmodelle abgeleitet werden könnten.

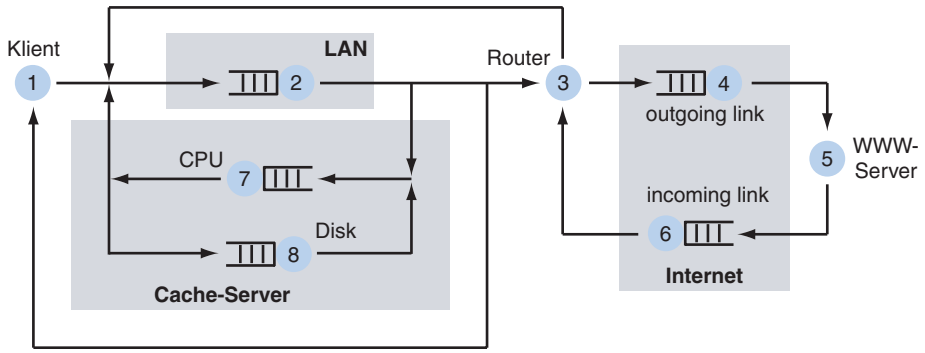


Abbildung 5.2: Warteschlangenmodell für einen lokalen Cache-Server [MeAl98]

Werden lediglich die Größen betrachtet, die aus den Logdateien eines Cache-Servers bekannt sind, ergibt sich eine deutliche Reduzierung des dargestellten Modells. Bei der Festlegung der auszuwertenden Informationen ist zu beachten, dass für eine Betrachtung der Übertragungsraten neben den Bearbeitungszeiten auch die Datenvolumen berücksichtigt werden müssen. Unmittelbar stehen aus den Logdateien folgende Informationen zur Verfügung:

- die Zwischenankunftszeiten der Requests,
- die Dauer der gesamten Bearbeitung eines Requests. Die Dauer setzt sich zusammen aus:
  - Wartezeit auf dem Cache-Server,
  - Bearbeitungszeit auf dem Cache-Server,
  - Übertragungszeit der Response (s. Kapitel 3.2.3).
- das übertragene Datenvolumen.

Ausgehend von diesen Informationen ergibt sich zunächst das in Abbildung 5.3 dargestellte Modell eines einzelnen Cache-Servers. Ergänzend sind in der Abbildung die Zeiten eingetragen, die allgemein bei der Betrachtung von Warteschlangen von Bedeutung sind.

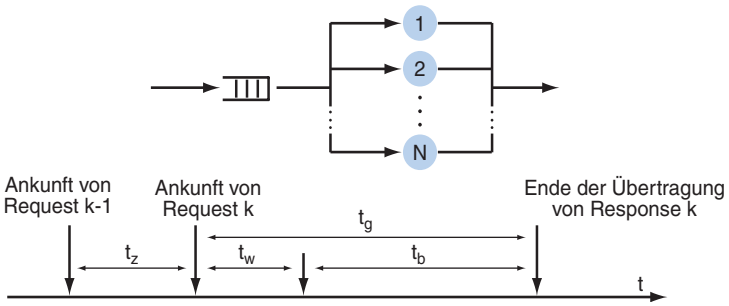


Abbildung 5.3: Vereinfachtes Warteschlangenmodell für einen Cache-Server

Das Modell besteht aus einer einzelnen Warteschlange sowie N Bedieneinheiten. Die Zahl der Bedieneinheiten entspricht den Aufträgen, die der Cache-Server parallel verarbeiten kann (s. Kapitel 2.2.2.4). Die Bedienzeit t_b enthält bereits die Übertragungszeit, die sich aus dem Volumen und der Datenrate ergibt, mit dem das Objekt oder die HTTP-Response übertragen wird.

Die tatsächliche Wartezeit t_w in der Warteschlange ist nicht bekannt. Aus der typischen Implementierung des TCP/IP-Stacks UNIX-basierter Betriebssysteme ergibt sich jedoch, dass eine Vernachlässigung von t_w zulässig ist. Auf einem UNIX-basierten System werden jedem TCP-Port zwei Warteschlangen für eingehende Requests zugewiesen. In der *incomplete connection queue* werden die Requests während des TCP-Verbindungsaufbaus (s. Kapitel 2.1.3.3) festgehalten. Nach erfolgreichem Verbindungsaufbau wandern die Requests in die *completed connection queue*, wo sie über den Systemaufruf `accept()` von der Anwendung für die weitere Bearbeitung ausgelesen werden. Die Größen beider Warteschlangen werden über Kernelparameter sowie über einen Funktionsparameter des Systemaufrufs `listen()` vorgegeben und sind ein bis zwei Größenordnungen kleiner als die Anzahl der Requests, die parallel verarbeitet werden können. Die Aufenthaltszeiten in diesen Warteschlangen bewegen sich nach [Ste99] bei nicht stark überlasteten Servern maximal im Bereich weniger Millisekunden und sind gegenüber den Übertragungszeiten der HTTP-Responses vernachlässigbar. Aus $t_b \gg t_w$ folgt daher die Näherung $t_g = t_b + t_w \approx t_b$.

Ausgehend von dieser Näherung wird bei der Implementierung des Simulationsmodells angenommen, dass eingehende Requests ohne Verzögerung an eine freie Bedieneinheit durchgereicht werden. Da die Warteschlange keine Requests zur späteren Verarbeitung aufnimmt, dient sie lediglich als Indikator für einen überlasteten Cache-Server: Sobald die Warteschlange einen oder mehrere Einträge enthält, müssen sämtliche N Bedieneinheiten belegt sein.

Für das dargestellte Modell gelten weiterhin die in der Warteschlangentheorie bekannten Beziehungen zwischen den Zwischenankunfts- und Bedienzeiten. So ergibt sich aus der mittleren Zwischenankunftsrate $\lambda = E[t_z]$ und der mittleren Bedienrate $\mu = E[t_g]$ die Stabilitätsbedingung für N Bedieneinheiten

$$\lambda < N \cdot \mu \tag{5.1}$$

5.1.2.2 Modell eines Cache-Verbundes

Für das Simulationsmodell eines Cache-Verbundes wird vorausgesetzt, das folgende Parameter variabel sind, jedoch während eines Simulationsdurchlaufs nicht verändert werden können:

- Anzahl der Partitionen im Cache-Verbund,
- Anzahl der Cache-Server in jeder Partition,
- Anzahl der Klienten bzw. lokalen Cache-Server,
- Zuordnung der Neighbors für die Klienten,
- Wahl der Protokolle zur Inter-Cache-Kommunikation,
- charakteristische Verkehrsparameter der Eingangslast.

Die Festlegung dieser Angaben erfolgt über eine Konfigurationsdatei. Da die aufgeführten Werte nicht interaktiv verändert werden können, sind für eine nähere Untersuchung dieser Parameter mehrere Simulationsdurchläufe mit verschiedenen Konfigurationen erforderlich. Für die Festlegung der optimalen Anzahl an Partitionen im Cache-Verbund bei vorgegebener Eingangslast und Anzahl Klienten sind z. B. mehrere Simulationsdurchläufe notwendig, in denen die konfigurierte Aufteilung der Cache-Server variiert wird.

Das Modell eines Cache-Verbundes ergibt sich in einem ersten Schritt aus der Parallelschaltung von C Cache-Servern nach Abbildung 5.3. Aus den in Kapitel 2.3 erläuterten Eigenschaften eines Cache-Verbundes sowie den in der Einleitung von Kapitel 4 genannten Forderungen an das zu implementierende Simulationsmodell ergeben sich folgende Ergänzungen:

- HTTP-Requests können innerhalb des Cache-Verbundes von einem Cache-Server an dessen Neighbors weitergeleitet werden. Daraus folgt eine Rückkopplung in dem Modell und eine Überlagerung der Eingangslast mit diesen Requests.
- Die Verteilung des Verkehrsaufkommens an die Cache-Server ist abhängig von den Klienten:
  - Die Last im Cache-Verbund wird durch eine begrenzte Anzahl von L Klienten erzeugt.
  - Die Klienten erzeugen unterschiedliche Verkehrsaufkommen.
  - Jedem Klienten l werden $c_l \leq C$ Neighbors im Cache-Verbund über die Konfiguration fest zugewiesen. Die Zuordnung zwischen Klienten und Neighbors ist determiniert.
  - Die Verteilung der HTTP-Requests eines Klienten an dessen Neighbors ist nicht determiniert, sondern hängt von den Zufallsfunktionen ab, mit denen Cache-Hits oder Cache-Misses generiert werden.
- Da das Senden von Requests durch die Klienten unabhängig von vorangegangenen, erfolgreich bearbeiteten Requests erfolgen soll (s. Kapitel 4.2.2.3), ist eine

Rückkopplung der ausgehenden Verkehrsflüsse an die Klienten nicht erforderlich. Das Modell kann deshalb durch ein offenes Warteschlangenmodell beschrieben werden.

- Um Ausfälle zu simulieren, sollen einzelne Cache-Server während der Simulation gezielt ein- und ausgeschaltet werden können.

Unter Berücksichtigung der aufgeführten Aspekte ergibt sich das in Abbildung 5.4 dargestellte Warteschlangenmodell.

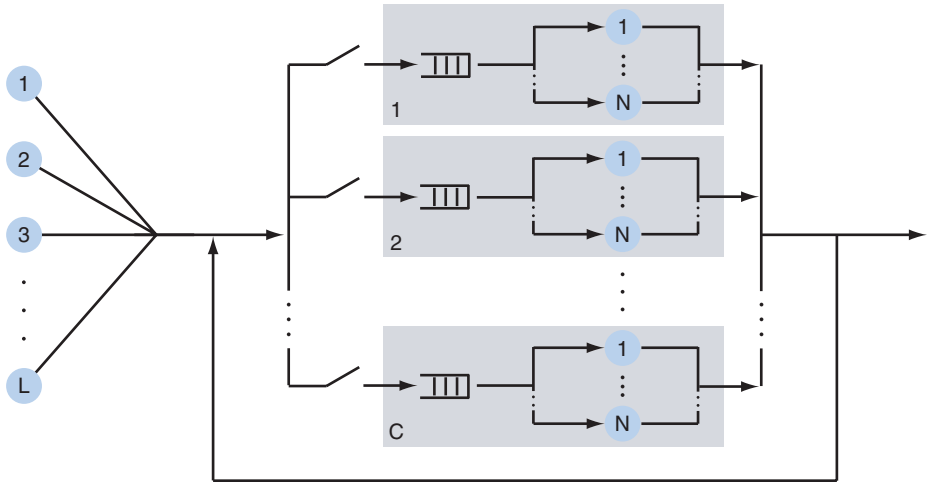


Abbildung 5.4: Warteschlangenmodell für einen Cache-Verbund

Der Versuch einer Klassifizierung des Modells nach Kendall [Ken51] liefert folgende Feststellungen:

- Nach Kapitel 4 werden exponentialverteilte Zwischenankunftszeiten angenommen.
- Die Verteilung der Bearbeitungszeiten wird nicht näher spezifiziert.
- Es stehen C Cache-Server zur Verfügung.
- Jeder Cache-Server kann N Aufträge gleichzeitig bearbeiten.
- Die Anzahl der Klienten ist auf L begrenzt. Von jedem Klienten kann eine nicht näher begrenzte Anzahl Aufträge gleichzeitig in dem Cache-Verbund verarbeitet werden.
- Jedem Klienten steht lediglich eine determinierte Teilmenge der Cache-Server zur Verfügung.

Da eine weiterführende analytische Behandlung des dargestellten Warteschlangenmodells im Rahmen dieser Arbeit nicht vorgesehen ist, wird auf eine den aufgeführten Feststellungen folgende Vereinfachung oder Modifikation verzichtet. Unabhängig davon liegt das Modell der folgenden Implementierung zugrunde.

5.2 Implementierung

5.2.1 Systemstruktur

Das Simulationsmodell lässt sich in fünf Komponenten unterteilen: Neben dem Lastgenerator, dessen Parameter sich aus der Modellierung in Kapitel 4 ergeben, zählen hierzu der Dispatcher für die Verteilung der generierten HTTP-Requests innerhalb des simulierten Cache-Verbundes, das Cache-Management, das die Zustände der einzelnen Cache-Server verwaltet, die Steuerung des Simulationsmodells sowie die Darstellung und Bedienung über eine grafische Oberfläche. Die grafische Oberfläche dient im Wesentlichen der Kontrolle des Simulationsverlaufs sowie der gezielten Ein- und Ausschaltung einzelner Cache-Server. Da die eigentliche Auswertung der Simulation in einem zweiten Schritt durch Aufbereitung der protokollierten Ereignisse erfolgt, kann alternativ auf die Nutzung der grafischen Oberfläche verzichtet werden. Hierdurch ist es möglich, mehrere längere Simulationen sequentiell im Batch-Betrieb durchzuführen. Der Zusammenhang zwischen den aufgeführten Komponenten ist aus dem Datenflussdiagramm in Abbildung 5.5 ersichtlich.

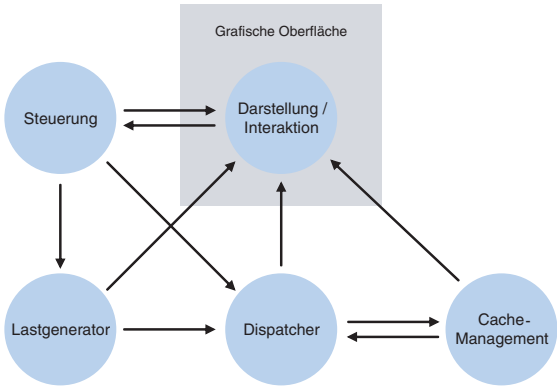


Abbildung 5.5: Komponenten des Simulationsmodells

Im Folgenden werden einzelne Aspekte bei der Implementierung der Komponenten hervorgehoben. Dabei steht die Umsetzung der in den vorangegangenen Kapiteln erarbeiteten Kommunikationsbeziehungen sowie die Darstellung der auftretenden Verkehrsflüsse im Vordergrund.

5.2.2 Lastgenerator

Mit dem Lastgenerator wird der Verkehrsfluss Out_{WWW} nachgebildet. Die hierfür notwendige Datenbasis enthält sämtliche der in Anhang C aufgeführten Ergebnisse aus den Modellierungen nach Kapitel 4. Der Lastgenerator wurde als eigenständiges Modul konzipiert, um den generierten Datenstrom bei Bedarf auch anderen Programmen, z. B. einem WWW-Klienten zur Generierung realer HTTP-Netzlast, zur Verfügung stellen zu können.

Datenbasis

Neben den Parametern der fünf betrachteten Datensätze kann der Lastgenerator bei der Initialisierung einen synthetischen Datensatz generieren. Über Angaben in der Konfigurationsdatei können dabei bestimmte Merkmale eines vorhandenen Datensatzes modifiziert werden. Das folgende Beispiel verwendet den Datensatz 01 als Grundlage, wobei eine höhere mittlere Ankunftsrate, zweifache Übertragungsdauern und eine neue Verteilung der Toplevel Domains angegeben wird:

```
WORKLOAD_MOD_BASE=01
WORKLOAD_MOD_INTER_ARRIVAL_TIME=450.0
WORKLOAD_MOD_DURATION_FACTOR=2.0
WORKLOAD_MOD_TLD_FRACTION={0.3, 0.5, 0.05, 0.05, 0.025, 0.025, 0.05}
```

Erzeugung der Verteilungsfunktionen

Steht eine gleichverteilte Zufallsvariable $u \sim U(0,1)$ zur Verfügung, kann daraus mit dem Ansatz $u = F_X(x)$ eine kumulative Verteilungsfunktion $F_X(x)$ erzeugt werden. Die notwendige inverse Transformation $x = F_X^{-1}(u)$ ist für die Exponential-, Pareto- und Weibullverteilung in Anhang B aufgeführt.

Die Generierung einer normalverteilten Zufallsvariablen ist mit diesem Verfahren nicht möglich, da für die kumulierte Normalverteilung keine Inverse angegeben werden kann. Hier wird der Algorithmus nach der Box-Muller-Methode [Knu81] verwendet, der eine standardnormalverteilte Zufallsvariable $x \sim \mathrm{N}(0,1)$ erzeugt. Durch die Umformung $x' = \mu + \sigma x$ wird daraus eine allgemeine normalverteilte Zufallsvariable $x \sim \mathrm{N}(\mu,\sigma)$. Entsprechend wird durch die Transformation $x' = e^{\mu_L + \sigma_L x}$ aus der standardnormalverteilten Zufallsvariablen $x \sim \mathrm{N}(0,1)$ eine lognormalverteilte Zufallsvariable $x' \sim \mathrm{LN}(\mu_L,\sigma_L)$.

Zufallsgenerator

Zufallsgeneratoren erzeugen in der Regel einen Wert u, der im Bereich $[0,1]$ gleichverteilt ist ($u \sim U(0,1)$). Bei der Auswahl eines geeigneten Zufallsgenerators muss ein Kompromiss zwischen der Periodendauer, der Auflösung sowie der Geschwindigkeit der Berechnung getroffen werden. Im Vorfeld dieser Arbeit wurden verschiedene Zufallsgeneratoren untersucht, von den genannten Aspekten war dabei die Auflösung von zentraler Bedeutung. Aus der Darstellung der komplementär kumulativen Verteilungsfunktion mit long-tail Eigenschaften in Abbildung 4.10 folgt für die Generierung von Zufallszahlen, dass bereits geringe Schwankungen von u zu signifikanten Unterschieden in den erzeugten Werten führen. Daher ist eine hohe Auflösung von u erforderlich, um einen breiten Ereignisraum auch über den Tail der Verteilungsfunktion abdecken zu können.

In dieser Arbeit wird ein Generator nach L'Ecuyer und Kelton verwendet [ESCK02] [LaKe00]. Neben einer hohen Auflösung und ausreichender Performance bietet die-

ser Zufallsgenerator verschiedene, so genannte Streams von Zufallszahlen. Indem jeder Zufallsfolge des Lastgenerators ein eigener Stream zugeordnet wird, können gezielt die Auswirkungen von Veränderungen der Workloads untersucht werden. So ist es möglich, in mehreren Simulationsläufen identische Folgen von Objektgrößen z. B. auf verschiedene Toplevel Domains zu verteilen oder mit unterschiedlichen Datenraten zu belegen.

5.2.3 Dispatcher

Der Dispatcher konzentriert und verteilt die simulierten Datenströme. Für die Verteilung verfügt der Dispatcher über ein Abbild der Struktur des Cache-Verbundes sowie der Zuordnungen zwischen übergeordneten und lokalen Cache-Servern. Die hierfür notwendigen Informationen werden bei der Initialisierung des Simulationsmodells aus der Konfigurationsdatei geladen. Änderungen des Cache-Verbundes während einer Simulation sind nicht vorgesehen.

Abbildung der Struktur des Cache-Verbundes

Folgende Optionen der Konfigurationsdatei enthalten die Anzahl der Partitionen, die Zuordnung der Toplevel Domains, die Anzahl der Cache-Server je Partition sowie die Neighbor-Beziehungen zwischen den Cache-Servern mit den jeweiligen Protokollen zur Inter-Cache-Kommunikation:

```
CACHE_PARTITIONS=3
CACHE_PARTITON_TLD={0, COM}
CACHE_PARTITON_TLD={1, DE}
CACHE_PARTITON_TLD={2, EDU, NET, NUM, ORG, OTHER}
CACHE_PARTITION_SIZE={5, 3, 2}      # partitions: 0-4 5-7 8-9
CACHE_SIBLING={0, 1, 2, ICP}        # 2 neighbors of cache 0
CACHE_SIBLING={0, 3, CD}            # 1 neighbor  of cache 0
CACHE_PARENT={0, 4, CD}             # 1 neighbor  of cache 0
CACHE_PARENT={1, 0, 2, 3, HTCP}     # 3 neighbors of cache 1
CACHE_SIBLING={2, 0, 1, 3, ICP}     # 3 neighbors of cache 2
```

Die Struktur des Cache-Verbundes wird über zwei zweidimensionale Arrays abgebildet, die die Neighbor-Beziehungen und die verwendeten Inter-Cache-Protokolle enthalten. Abbildung 5.6 zeigt den Aufbau der Arrays mit den Einträgen aus der oben angegebenen Konfiguration.

Zuordnung zwischen Klienten und übergeordneten Cache-Servern

Da das Simulationsmodell die Betrachtung von bis zu 250 lokalen Cache-Servern gestatten soll, können die Neighbor-Beziehungen und Inter-Cache-Protokolle zwischen lokalen und übergeordneten Cache-Servern nicht über einzelne Optionen in der Konfigurationsdatei angegeben werden. Die Anzahl der Cache-Server je Partition wurde bereits durch die oben genannte Option `CACHE_PARTITION_SIZE` festgelegt. Mit der Option

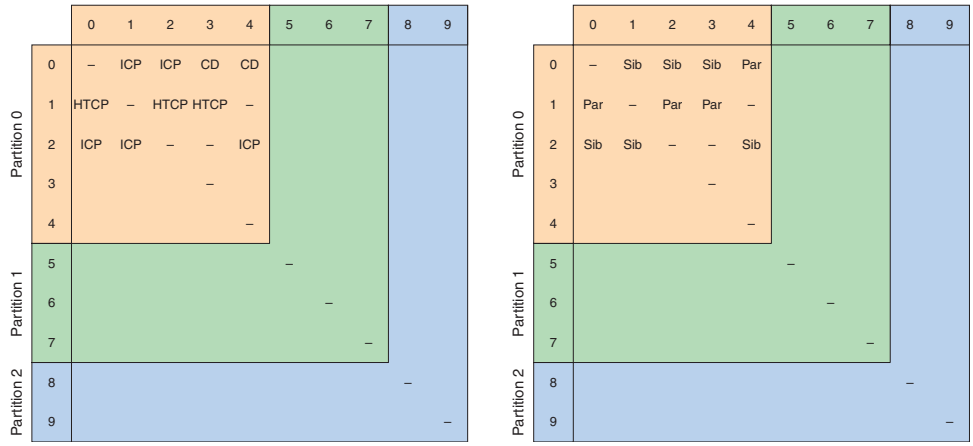


Abbildung 5.6: Neighbor-Beziehungen und Inter-Cache-Kommunikation

```
CACHE_PARTITION_ASSIGN={0, 3, 2}
```

werden z. B. jedem lokalen Cache-Server 3 übergeordnete Cache-Server aus Partition 0 zugewiesen. Die Festlegung, wie die Cache-Server innerhalb einer Partition den lokalen Cache-Servern zugeordnet werden, erfolgt über einen Offset als drittem Parameter. Aus diesen Parametern lässt sich ein Algorithmus formulieren, mit dem alle Zuordnungen dargestellt werden können. Für die Erläuterung des Algorithmus werden zunächst die Definitionen aus Kapitel 3.2.5 mit geringfügigen – programmiertechnisch bedingten – Modifikationen und Ergänzungen übernommen:

- L: Anzahl der lokalen Cache-Server.
- P: Anzahl der Partitionen im übergeordneten Cache-Verbund.
- c_p mit $p = \{0,\ldots,P-1\}$: Anzahl der übergeordneten Cache-Server in Partition p.
- o_p mit $p = \{0,\ldots,P-1\}$: Offset in Partition p.
- n_p mit $p = \{0,\ldots,P-1\}$: Anzahl der übergeordneten Cache-Server (Neighbors) in Partition p, die von einem lokalen Cache-Server befragt werden.

Die Zuordnung zwischen den L lokalen Cache-Servern und deren n_p übergeordneten Cache-Servern aus Partition p wird über ein dreidimensionales Array `neighbor_id[L][P][n_p]` abgebildet, wobei sich die Einträge in dem Array nach folgendem Algorithmus ergeben:

```
for ( l = 0; l < L; l++ )          # lokale Cache-Server
  for ( p = 0; p < P; p++ )        # Partitionen
    for ( a = 0; a < n[p]; a++ )   # Cache-Server in Partition p
      neighbor_id[l][p][a] = ( l * o[p] + a ) % c[p];
```

Aus den oben beispielhaft angegebenen Optionen ergeben sich für die Partition $p = 0$ folgende Werte:

- $c_0 = 5$ Cache-Server,
- $n_0 = 3$ Neighbors je lokalem Cache-Server,
- Offset $o_0 = 2$.

Mit diesen Angaben werden dem lokalen Cache-Server $l = 0$ die übergeordneten Cache-Server 0, 1 und 2, dem lokalen Cache-Server $l = 3$ die übergeordneten Cache-Server 4, 0 und 1 aus dieser Partition zugewiesen.

Inter-Cache-Kommunikation

Die Inter-Cache-Kommunikation wird gemäß den in den Spezifikationen festgelegten Regeln nachgebildet. Bei der Verwendung von ICP und HTCP gehen sämtlichen HTTP-Requests ICP- bzw. HTPC-Requests an alle konfigurierten Neighbors der entsprechenden Partition voraus. Um das maximal mögliche Verkehrsaufkommen zu betrachten, werden Verluste von ICP- oder HTPC-Requests nicht berücksichtigt.

Das Datenvolumen und die Häufigkeit für den Austausch von Cache Digests wird für den gesamten Cache-Verbund durch folgende Optionen gesetzt:

```
ICC_CD_FREQUENCY=3600       # in seconds
ICC_CD_SIZE=2097152         # in bytes
```

Die Datenrate, mit der ein Cache-Digest übertragen wird, entspricht der eines Cache-Hit aus der jeweiligen Partition.

Die Angabe der Protokolle zur Inter-Cache-Kommunikation zwischen lokalen und übergeordneten Cache-Servern erfolgt über die Optionen

```
ICC_LOCAL_PROTOCOL={CD, ALL}
ICC_LOCAL_PROTOCOL={ICP, 0-2, 17}
```

Mit der ersten Zeile wird festgelegt, dass sämtliche lokalen Cache-Server zur Inter-Cache-Kommunikation mit den übergeordneten Cache-Servern Cache Digests nutzen. Anschließend werden die Ausnahmen dieser Regel aufgeführt: die lokalen Cache Server 0, 1, 2 und 17 verwenden ICP. Es ist nicht vorgesehen, dass ein lokaler Cache-Server unterschiedliche Protokolle zur Inter-Cache-Kommunikation mit verschiedenen übergeordneten Cache-Servern verwendet.

5.2.4 Cache-Management

Das Cache-Management verwaltet die Zustände aller übergeordneten Cache-Server. Der Zustand eines Cache-Servers wird in dem Simulationsmodell durch die Anzahl momentan bearbeiteter Responses, die Summendatenrate aller Responses sowie ein Status-Flag definiert. Der gesamte Ablauf einer Simulation erfolgt ereignis- und nicht zeitgesteuert, wobei als Ereignis die Zustandsänderung eines Cache-Servers angesehen wird. Daraus folgt, dass der Fortschritt der Simulation durch eintreffende Requests, das Ende einer Übertragung sowie über den Beginn und das Ende einer Störung bestimmt wird.

Verwaltung der Zustandsänderungen

Die Zustandsänderungen werden für jeden Cache-Server über eine verkettete Liste verwaltet. Die Knoten einer Liste enthalten alle anstehenden Zustandsänderungen, die Wurzel weist auf den Knoten mit der nächsten Änderung. Abbildung 5.7 illustriert die folgende Erläuterung der Implementierung:

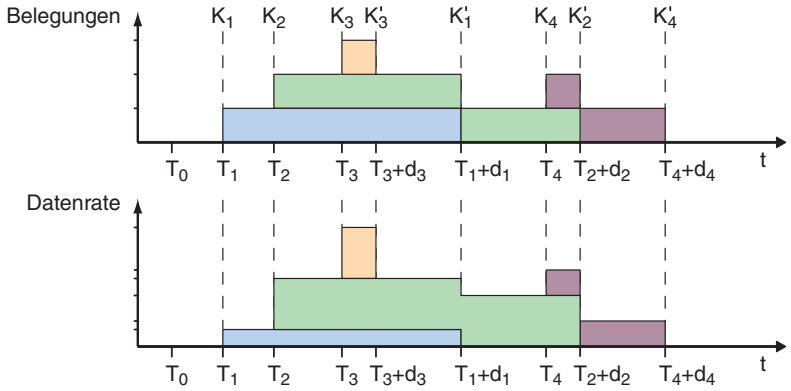


Abbildung 5.7: Implementierung Cache-Management

1. Zum Zeitpunkt T_0 wird kein HTTP-Response von dem Cache-Server verarbeitet, die Liste ist leer.

2. Mit dem Eintreffen eines Requests zum Zeitpunkt T_1 mit der entsprechenden Übertragungsdauer d_1 für die Response werden zwei Knoten in die Liste eingeführt: der Knoten K_1 enthält $B = 1$ Belegungen und die Datenrate $R = r_1$, ein weiterer Knoten K'_1 stellt die Zustandsänderung zum Zeitpunkt $T_1 + d_1$ auf $B = 0$ Belegungen und die Datenrate $R = 0$ dar. Die Wurzel der Liste weist auf den Knoten K_1.

3. Ein zweiter Request zum Zeitpunkt T_2 fügt einen Knoten K_2 vor und einen Knoten K'_2 nach dem Knoten K'_1 ein. Der Knoten K_2 enthält die Werte aus dem Knoten K_1 erhöht auf $B = 2$ Belegungen und die Datenrate $R = r_1 + r_2$. Alle Knoten zwischen K_2 und K'_2 werden ebenfalls um eine Belegung und die Datenrate r_2 erhöht. Die Wurzel der Liste weist auf den Knoten K_2.

4. Der Knoten K_3 enthält $B = 3$ Belegungen und die Datenrate $R = r_1 + r_2 + r_3$. Da die Übertragung des Objekts noch vor der nächsten Zustandsänderung, die mit dem Knoten K'_2 signalisiert wird, beendet ist, wird der Knoten K'_3 vor K'_2 eingefügt. Der Knoten K'_3 enthält entsprechend $B = 2$ Belegungen und die Datenrate $R = r_1 + r_2$. Die Wurzel der Liste weist auf den Knoten K_3.

Die weiteren Requests werden nach dem genannten Verfahren sukzessive bearbeitet. Es ist zu beachten, dass die Belegungen diskrete und die Datenraten kontinuierliche Werte annehmen können.

Ablaufsteuerung

Um den gesamten zeitlichen Ablauf geeignet zu steuern, führt der Dispatcher eine weitere verkettete Liste, die als Knoten die Wurzeln aller verketteten Listen aus dem Cache-Management enthält. Über die Angaben in den Knoten dieser Liste verfügt der Dispatcher über den aktuellen Zustand aller Cache-Server. Da die Liste ebenfalls in einer zeitlichen Ordnung geführt wird, zeigt die Wurzel auf den Zeitpunkt der nächsten Zustandsänderung bzw. des nächsten Verarbeitungsschritts. Darüber hinaus kann der Dispatcher, nachdem er die Zwischenankunftszeit bis zum nächsten eintreffenden Request ermittelt hat, sämtliche abzuarbeitenden Zustandsänderungen unmittelbar abrufen. Bei auftretenden Störungen werden alle Knoten der Liste des entsprechenden Cache-Servers gelöscht und dessen Zustandswerte auf 0 gesetzt.

Um auch Zwischenankunftszeiten unterhalb von 1 ms geeignet dargestellen zu können, werden sämtliche zeitbezogenen Werte der Simulation mit einer Auflösung von 100 µs berechnet.

Grenzwerte

Um Lastgrenzen der Cache-Server zu definieren, können jeweils zwei Parameter für die momentan bearbeiteten Responses und die Summendatenrate angegeben werden. Mit folgenden Angaben werden die Parameter für alle Cache-Server gesetzt:

```
CACHE_RESPONSES_WARNING=512      # HTTP-Responses
CACHE_RESPONSES_CRITICAL=1024    # HTTP-Responses
CACHE_BANDWIDTH_WARNING=60       # in MBit/second
CACHE_BANDWIDTH_CRITICAL=80      # in MBit/second
```

Bei Erreichen der Schwellwerte für die Warnungen werden lediglich die entsprechenden Anzeigen in der grafischen Oberfläche gelb eingefärbt. Die Schwellwerte für die kritischen Zustände können jedoch nicht überschritten werden. Die Annahme eines neuen HTTP-Requests ist erst dann wieder möglich, wenn die Verarbeitung einer HTTP-Response abgeschlossen wird. Cache-Server, deren HTTP-Responses oder Datenraten einen kritischen Schwellwert erreicht haben, werden in der grafischen Oberfläche rot markiert. Gleichzeitig wird durch ein auf `CRITICAL` gesetztes Aktivitäts-Flag dem Dispatcher mitgeteilt, dass der Cache-Server keine weiteren Requests zur Verarbeitung entgegennehmen kann.

5.2.5 Grafische Oberfläche und Steuerung

Das grafische Interface des Simulationsmodells wurde mit Hilfe der Qt-Bibliothek [Tro02] realisiert. Die Oberfläche dient im Wesentlichen der Überwachung des Simulationsverlaufs sowie der interaktiven Einführung von Störungen einzelner Cache-Server. Abbildung 5.8 zeigt die einzelnen Fenster der Oberfläche. Die Darstellung der aktuellen Zustände aller Cache-Server in dem Fenster Cache Hierarchy Status zeigt, dass der Cache-Server mit der ID 4 den kritischen Schwellwert von 1024 be-

arbeiteten HTTP-Responses erreicht hat. Die Cache-Server 6 und 7 sind ebenfalls rot markiert. Die Angabe von 0 bearbeiteten HTTP-Responses weist jedoch darauf hin, dass beide Cache-Server nach Störungen ausgefallen sind. Durch Markieren der Auswahlfelder in der linken Spalte des Fensters können die Cache-Server wieder aktiviert werden. HTTP-Requests, die an eine Partition gerichtet sind, in der aufgrund Störung oder Überlastung kein Cache-Server zur Verfügung steht, werden in der Zeile Rejected in dem Fenster WWW Flow protokolliert. Die weiteren Fenster zeigen den aktuellen Stand der Verkehrsflüsse und enthalten Informationen über den Fortschritt der Simulation.

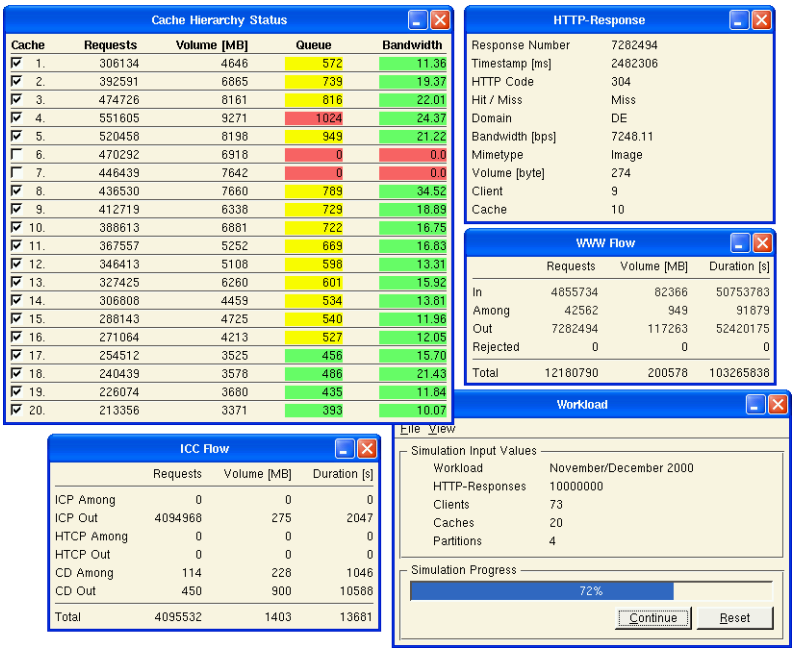


Abbildung 5.8: Fenster der grafischen Oberfläche

Um Störungen auch zu definierten Zeitpunkten nachbilden zu können, stehen vier Optionen zur Verfügung:

```
CACHE_FAILURE_REQUESTS={5, 9800000}
CACHE_RECOVER_REQUESTS={5, 9900000}
CACHE_FAILURE_TIME={3, 600000}          # in 100 us
CACHE_RECOVER_TIME={3, 1800000}         # in 100 us
```

In dem Beispiel steht der Cache-Server 5 während der simulierten HTTP-Responses 9800000 – 9900000 nicht zur Verfügung, der Cache-Server 3 während der simulierten Zeitspanne von der 60. – 180. Sekunde.

5.3 Bewertung des Simulationsmodells

5.3.1 Performance

Eine Abschätzung der Performance des Simulationsmodells wird durch Benchmarking verschiedener Konfigurationen ermittelt. In 45 verschiedenen Simulationsläufen werden die mittlere Zwischenankunftszeit $1/\lambda$ und die Datenraten r variiert, wobei jeweils die Zeit für die Generierung von 100.000 – 10.000.000 HTTP-Responses gemessen wird. Die in Abbildung 5.9 dargestellten Ergebnisse wurden auf einem Linux-PC mit einem Prozessor Intel Pentium 3 933 MHz gerechnet. Die Messungen zeigen zunächst einen linearen Verlauf zwischen der Anzahl simulierter HTTP-Responses und der Dauer der Simulation. Eine Verringerung der mittleren Zwischenankunftszeiten oder ein Anstieg der Datenraten verändert die Simulationsdauer nur geringfügig. Umgekehrt führen jedoch eine höhere Ankunftsrate und eine längere Übertragungsdauer zu einer deutlich geringeren Performance. Die Ursache hierfür liegt in dem durch permanentes Einfügen neuer und Aktualisieren vorhandener Zustandsänderungen erhöhten Verwaltungsaufwand des Cache-Management.

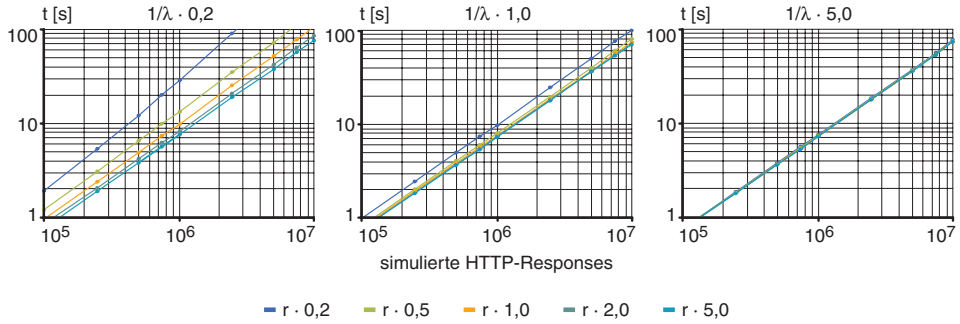


Abbildung 5.9: Performance des Simulationsmodells

Bei unveränderter Konfiguration arbeitet das Simulationsmodell mit ca. 130.000 simulierten HTTP-Responses/s, bei fünffacher Übertragungsdauer und fünffacher Ankunftsrate sinkt die Rate auf ca. 22.000 HTTP-Responses/s. In einer hier nicht dargestellten Untersuchung wurden zusätzlich verschiedene Konfigurationen des Cache-Verbundes mit jeweils 4, 8, 16 und 32 Cache-Servern untersucht. Dabei zeigte sich, dass die Ergebnisse weitgehend unabhängig von der Größe und Struktur des simulierten Cache-Verbundes sind.

5.3.2 Qualität der Ergebnisse

Um die Qualität des Lastgenerators geeignet beurteilen zu können, wurden in zehn Simulationsläufen mit unterschiedlichen Startvektoren des Zufallsgenerators jeweils 10.000.000 HTTP-Responses mit den Parametern aus dem unmodifizierten Datensatz 01 untersucht. Aus den in Tabelle 5.1 aufgeführten Ergebnissen können

zunächst die charakteristischen Mittelwerte des generierten Datenstroms berechnet werden.

HTTP-Responses	10.000.000
simulierter Zeitraum [ms]	34.215.468
Datenvolumen [MByte]	160.072
gesamte Übertragungsdauer [ms]	28.738.003.547

Tabelle 5.1: Parameter der simulierten Workload

Aus den Werten ergibt sich eine mittlere Zwischenankunftszeit von $1/\lambda_{sim} = 3{,}4215$ ms, eine mittlere Übertragungsdauer von $\bar{d}_{sim} = 2{,}8738$ s und eine mittlere Übertragungsrate von $\bar{R} = 39{,}24$ MBit/s. Der Vergleich mit den in Kapitel 4.2 berechneten Werten zeigt sehr gute Näherungen für die Zwischenankunftszeit und die Übertragungsrate. Demgegenüber weicht die mittlere Übertragungsdauer von dem gemessenen Wert $\bar{d}_{HvStd} = 3{,}2691$ s um 12,5% ab. Diese Abweichung erklärt sich zum einen dadurch, dass in dem Simulationsmodell die maximal zulässige Übertragungsdauer auf einen Tag begrenzt wird. In einer nachträglichen Untersuchung konnten jedoch im Datensatz 01 Übertragungen mit einer Dauer von über 36 Stunden festgestellt werden, die maßgeblich zu der Verschiebung des Mittelwertes beitragen. Da die alten DFN-Cache-Server täglich neu gestartet wurden, tritt eine derart hohe Übertragungsdauer ausschließlich im Datensatz 01 auf. Ein weiterer Grund für die Abweichung der simulierten Übertragungsdauer ist in einer unzureichenden Qualität der Approximationen für niedrige Datenraten zu vermuten.

In Abbildung 5.10 links werden die Verteilungen der gemessenen und simulierten Übertragungsdauer verglichen. Da sich die Übertragungsdauer aus Objektgrößen und Datenraten zusammensetzen, stellt diese Verteilung einen guten Indikator für die Qualität des generierten WWW-Verkehrs dar. Der Vergleich der Verteilungen zeigt, dass die Simulation gute Näherung bis ca. 90% der Übertragungsdauer liefert. Bei höherer Übertragungsdauer zeigt sich jedoch eine Verschiebung, die auf dieselben Ursachen zurückzuführen ist wie die oben erläuterte Abweichung der Mittelwerte.

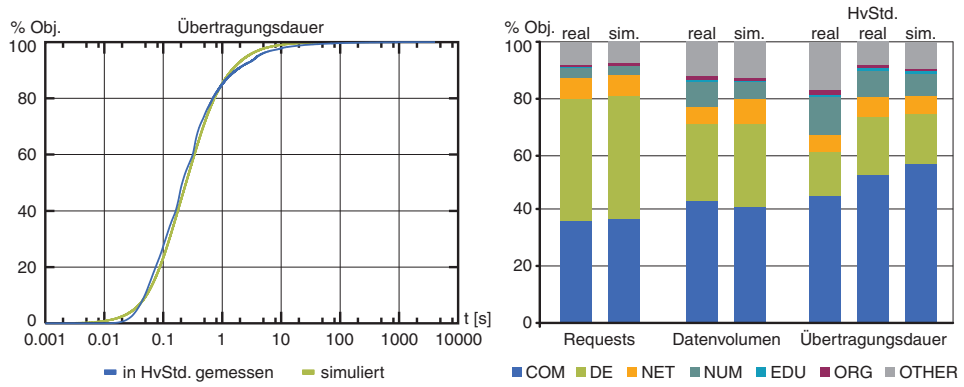


Abbildung 5.10: Vergleich realer und simulierter WWW-Verkehr

Mit einer weiteren Untersuchung werden, analog zu der Betrachtung in Kapitel 4, die Verteilungen von HTTP-Responses, Datenvolumen und Übertragungsdauer nach Toplevel Domains untersucht. Ein Vergleich mit den realen Messungen aus Abbildung 4.7 zeigt lediglich geringe Abweichungen. Da in Abbildung 4.7 die Übertragungsdauer für den vollständigen Datensatz betrachtet wurde, enthält Abbildung 5.10 rechts ergänzend die entsprechende Verteilung für die Hauptverkehrsstunde.

5.3.3 Funktionalität und Stabilität

In den folgenden Untersuchungen wird geprüft, ob sich die simulierten Datenströme und das Simulationsmodell gemäß dem Entwurf und der Spezifikation in Kapitel 5.2 verhalten. Hierfür wird ein Cache-Verbund aus zwei Partitionen mit je zwei Cache-Servern betrachtet. Die erste Partition wird den Toplevel Domains `com` und `num` zugeordnet, die zweite Partition den verbleibenden Toplevel Domains. Nach den gemessenen Werten aus Abbildung 5.10 sind ungefähr gleiche Anteile bezüglich des übertragenen Datenvolumens und ein Verhältnis von 2 zu 3 bezüglich der bearbeiteten HTTP-Responses zu erwarten.

Die erwarteten Verhältnisse werden mit den Ergebnissen in Abbildung 5.11 bestätigt. Nach Ablauf von einer Stunde ist das übertragene Datenvolumen auf alle vier Cache-Server annähernd gleich verteilt, während von Cache-Servern der Partition `com` und `num` lediglich 40% der HTTP-Responses verarbeitet werden.

Bei genauer Betrachtung der Datenvolumen in Abbildung 5.11 lassen sich Auswirkungen der long-tail-verteilten Objektgrößen erkennen. Zwischen der 20. – 40. Minute sind signifikante Unterschiede in den bearbeiteten Datenvolumen zwischen allen vier Cache-Servern zu sehen, wogegen die Volumen zwischen der 50. – 55. Minute nahezu identisch sind. Gegen Ende der Simulation steigt das Volumen auf zwei Cache-Servern deutlich an. Diese Treppeneffekte werden durch die für long-tail Verteilungen typischen, vereinzelt auftretenden Objekte mit sehr hohem Datenvolumen verursacht.

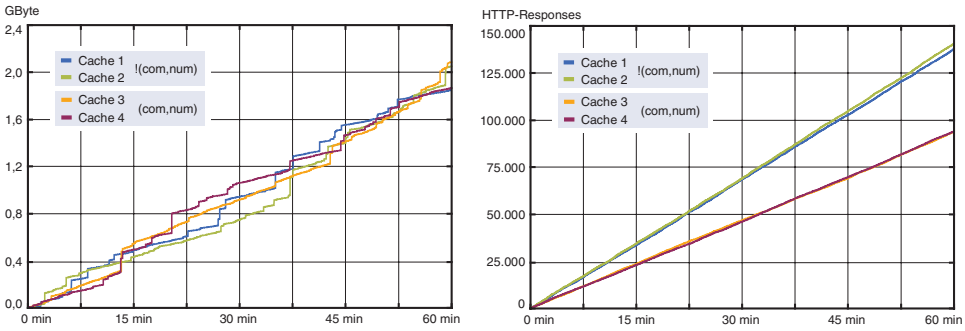


Abbildung 5.11: Vergleich Datenvolumen mit HTTP-Responses

Deutlicher können diese Effekte, die auch bei einer erforderlichen Stabilitätsbetrachtung von Bedeutung sind, an den Übertragungsraten untersucht werden. Abbildung 5.12 zeigt die Datenrate des Datenstroms, der von einem Cache-Server an dessen Klienten gesendet wird. In der linken Abbildung ist der gesamte Zeitraum von 9,5 Stunden eingetragen, in der rechten Abbildung ein Ausschnitt von der 8. – 9. Stunde. Zusätzlich enthalten beide Diagramme die Mittelwerte über 30 Sekunden und 5 Minuten.

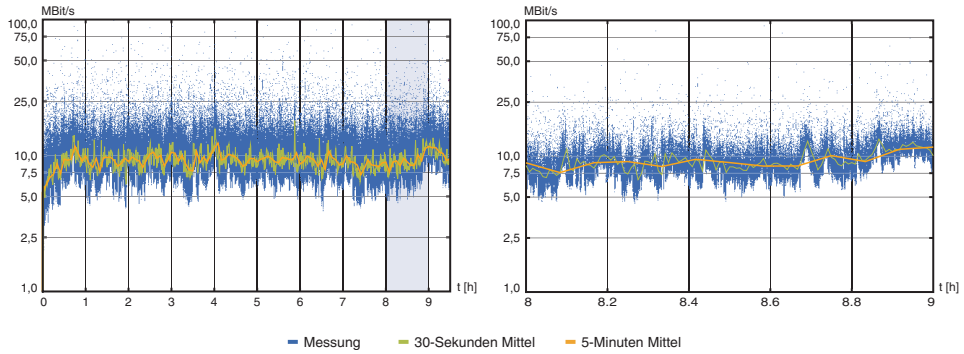


Abbildung 5.12: Betrachtung der Datenraten

Beide Diagramme zeigen die auch aus praktischen Messungen bekannten hohen Schwankungen der Datenrate. Dabei ist zu beachten, dass in der Simulation mit konstanten mittleren Datenraten für jedes Objekt gerechnet wird. Die auf Netzen typischen Schwankungen innerhalb einzelner Übertragungen, die in der Regel eine noch höhere Streuung der Summendatenrate ergeben, werden nicht nachgebildet.

Die Mittelwerte geben einen Hinweis auf den Verlauf bzw. die Stabilität der Simulation. Aus der eigentlichen Messreihe ist nicht zu erkennen, ob oder nach welcher Zeit die Simulation einen stabilen Zustand erreicht. So zeigt das über 5 Minuten gemessene Mittel in dem rechten Diagramm einen stetigen Anstieg und gibt damit einen scheinbaren Hinweis darauf, dass auch gegen Ende der Simulation noch kein stabiler Zustand erreicht wird. Das linke Diagramm in Abbildung 5.12 zeigt jedoch, dass die Summendatenrate in dem folgenden Abschnitt deutlich sinkt. Hier zeigen sich die Auswirkungen der long-tail-verteilten Übertragungsdauer, die in einer notwendigen Betrachtung ausreichend langer Zeiträume resultiert.

Anhand eines letzten Beispiels werden der Einschwingvorgang sowie die transienten Vorgänge bei Störungen in einem Cache-Verbund aus drei Cache-Servern untersucht. Der Lastgenerator erzeugt durch eine modifizierte geringere mittlere Ankunftsrate eine höhere Last, so dass im Mittel ca. 1.200 HTTP-Responses bearbeitet werden. Der Cache-Verbund ist nicht partitioniert, jedem lokalen Cache-Server sind alle drei Cache-Server als Neighbors zugeordnet. Zu erwarten ist eine auf alle Cache-Server gleich verteilte Last, wobei die mittlere Anzahl bearbeiteter HTTP-Responses um den Wert 400 schwankt.

Abbildung 5.13 zeigt die Simulation über einen Zeitraum von zwei Stunden. Dem Einschwingvorgang in den ersten 30 Minuten folgt eine Störung auf dem Cache-Server 2, die von den verbleibenden zwei Cache-Servern 1 und 3 abgefangen wird. Mit der zweiten Störung wird die gesamte Last auf den Cache-Server 1 konzentriert. Da dessen Maximum für gleichzeitig zu bearbeitende HTTP-Responses auf den Wert 1024 gesetzt ist, kommt es zu einer Überlastsituation, in der ein Teil der HTTP-Requests abgewiesen wird. Mit dem Einschalten von Cache-Server 3 nach 60 Minuten kann die gesamte Last wieder bearbeitet werden. Da eine lastabhängige Verteilung in dem Simulationsmodell nicht implementiert ist, werden die HTTP-Requests auf beide Cache-Server gleich verteilt, wodurch sich die Entlastung von Cache-Server 1 verzögert. In der Praxis wäre eine vorübergehende priorisierte Lenkung der HTTP-Requests auf Cache-Server 3 von Vorteil, um Cache-Server 1 schneller zu entlasten. So zeigt sich mit dem Ende der zweiten Störung nach 90 Minuten, dass durch die zwei gleichzeitig wieder eingeschalteten Cache-Server 1 und 2 die Last wesentlich schneller von Cache-Server 3 genommen wird.

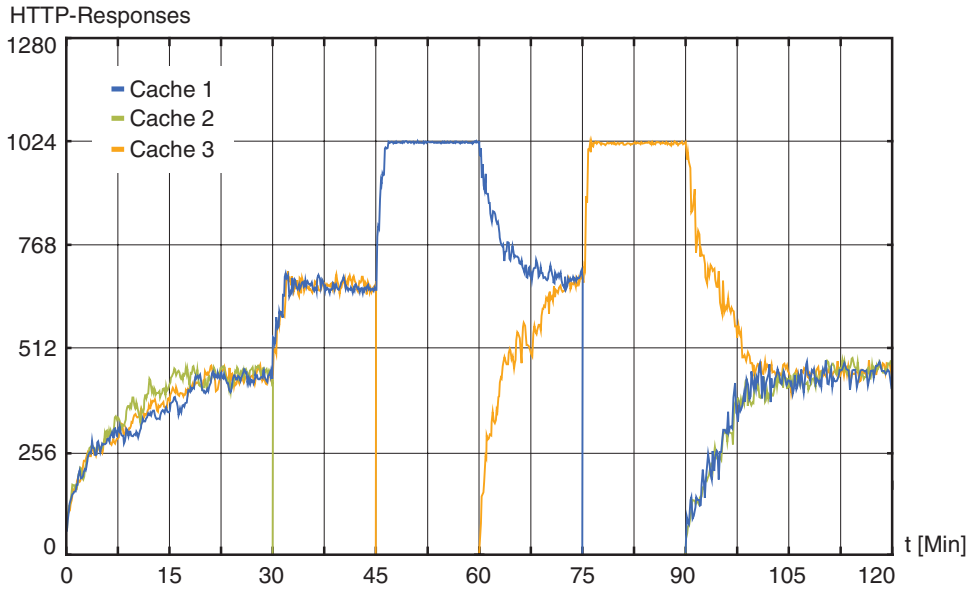


Abbildung 5.13: Simulation von Störungen in einem Cache-Verbund

In den aufgeführten Beispielen wurden Zeiträume von bis zu 9 Stunden betrachtet. Dabei ist zu beachten, dass die Simulationen mit Parametern aus der Hauptverkehrsstunde durchgeführt werden. Nach Abbildung 4.13 weist der Tagesverlauf der Hauptverkehrsstunde eine deutliche Charakteristik auf. Aus dieser Charakteristik folgt, dass die Bedingungen, mit denen die Simulationen durchgeführt werden, in der Realität lediglich innerhalb eines begrenzten Zeitraumes gültig sind. Außerhalb dieses Zeitraumes gelten geringere Anforderungen, so dass die Auswirkungen von auch nach dem Einschwingvorgang kontinuierlich steigenden Verkehrslasten in dem Simulationsmodell durch geringeres Verkehrsaufkommen außerhalb der Hauptverkehrsstunde in der Praxis aufgehoben werden. Die daraus resultierende

Frage nach dem Ende des Einschwingvorgangs und dem Beginn des auszuwertenden stabilen Zeitraumes ergibt sich daher aus einer grafischen Auswertung oder über den Vergleich der Mittelwerte über verschiedene Zeiträume. Aufgrund der long-tail-verteilten Übertragungsdauer ist dabei die Betrachtung ausreichend langer Zeiträume zu beachten.

5.3.4 Beispiel: Untersuchung von Cache-Verbünden

Die Anzahl der Cache-Server in einem Cache-Verbund richtet sich in erster Linie nach dem zu erwartenden Verkehrsaufkommen. So wäre eine wesentliche Leistungssteigerung in den DFN-Cache-Verbünden nur durch Hinzufügen weiterer Cache-Server möglich gewesen. Mit steigender Anzahl Cache-Server ergibt sich zunehmend die Frage nach einer günstigen Partitionierung des Cache-Verbundes, die sich in einer optimalen Zuordnung der lokalen Cache-Server und einer Minimierung der Inter-Cache-Kommunikation darstellt. Mit dem folgenden abschließenden Beispiel wird die Verkehrsverteilung in Cache-Verbünden unterschiedlicher Konfiguration bei konstanter Eingangslast verglichen. Dabei wird sowohl auf die in Kapitel 3.2 erarbeiteten Gleichungen als auch auf das Simulationsmodell zurückgriffen.

Betrachtet werden Cache-Verbünde aus 4, 8 und 16 Cache-Servern mit jeweils 1, 2 oder 3 Partitionen. Aus der Nebenbedingung, dass jede Partition mindestens zwei Cache-Server enthalten soll, folgt, dass mit vier Cache-Servern maximal zwei Partitionen gebildet werden können. Weiterhin werden jedem lokalen Cache-Server zwei übergeordnete Cache-Server aus jeder Partition zugewiesen. Der Offset zur Verteilung der lokalen Cache-Server wird ebenfalls auf den Wert 2 gesetzt (Kapitel 5.2.3).

Die vorgegebene Zuordnung der Toplevel Domains auf die Partitionen richtet sich nach dem übertragenen Datenvolumen. Ein Algorithmus für die optimale Zuordnung der Toplevel Domains bei vorgegebener Anzahl Cache-Server ergibt sich aus dem bekannten Rucksack-Problem, dessen Lösung u. a. für die hier vorliegenden einfachen Randbedingungen (Anteile als Ganzzahlen darstellbar, geringe Anzahl der Werte) aus [Sed90] übernommen werden kann. Die Konfigurationen der im Weiteren untersuchten Cache-Verbünde sind in Tabelle 5.2 zusammengefasst.

		Cache-Server		
		4	**8**	**16**
Partitionen	**1**	• 4 Server alles	• 8 Server alles	• 16 Server alles
	2	• 2 Server `com, num` • 2 Server !(`com, num`)	• 4 Server `com, num` • 4 Server !(`com, num`)	• 8 Server `com, num` • 8 Server !(`com, num`)
	3	–	• 3 Server `com` • 2 Server `de` • 2 Server !(`com, de`)	• 6 Server `com` • 5 Server `de` • 5 Server !(`com, de`)

Tabelle 5.2: Partitionierung und Zuordnung der Toplevel Domains

Das Verkehrsaufkommen zur Inter-Cache-Kommunikation kann mit den Gleichungen aus Kapitel 3.2.5 angegeben werden. Dabei wird vorausgesetzt, dass jedem lokalen Cache-Server zwei Cache-Server aus jeder Partition zugewiesen werden und dass die Cache-Server innerhalb jeder Partition untereinander vollständig vermascht als Neighbors konfiguriert sind.

Den Berechnungen liegen die Angaben für den Datensatz 01 zugrunde, d. h. die Klienten bestehen aus 73 lokalen Cache-Servern. Das Volumen eines Cache-Digests umfasst 2 MByte, die Cache-Digests werden stündlich ausgetauscht. Für den betrachteten Zeitraum von 28 Tagen folgt daraus, dass jeder Cache-Server 672 mal Cache-Digests von seinen Neighbors abruft. Tabelle 5.2 zeigt das Verkehrsaufkommen zur Übertragung von Cache-Digests an die lokalen Cache-Server, dass nach Gleichung 3.9 linear von der Anzahl der Partitionen und der Anzahl Neighbors abhängt.

Partitionen		
1	**2**	**3**
98.112 Requests 191,6 GByte	196.224 Requests 383,3 GByte	294.336 Requests 574,9 GByte

Tabelle 5.3: Cache Digests zwischen lokalen und übergeordneten Cache-Servern

Durch die nach Gleichung 3.10 quadratische Abhängigkeit von der Anzahl Neighbors wird das Verkehrsaufkommen zur Übertragung von Cache-Digests innerhalb des Cache-Verbundes wesentlich von der Anzahl Cache-Server innerhalb einer Partition bestimmt. Tabelle 5.4 zeigt die für die betrachteten Cache-Verbünde berechneten Werte.

Partitionen	**Cache-Server**		
	4	**8**	**16**
1	8.064 Requests 15,8 GByte	37.632 Requests 73,5 GByte	161.280 Requests 315,0 GByte
2	1.344 Requests 2,6 GByte	16.128 Requests 31,5 GByte	75.264 Requests 147,0 GByte
3	–	10.752 Requests 21,0 GByte	47.040 Requests 91,9 GByte

Tabelle 5.4: Cache Digests innerhalb des übergeordneten Cache-Verbundes

Eine entsprechende Berechnung kann auch für die Verwendung von ICP zur Inter-Cache-Kommunikation angegeben werden. Aus der einfachen Beziehung nach Gleichung 3.5 folgt mit $n = 2$ und $R_{Out_{\mathrm{WWW}}} = 287.872.746$ Responses aus Tabelle 3.14 direkt die Anzahl der an die lokalen Cache-Server übertragenen ICP-Replies $R_{Out_{\mathrm{ICP}}} = 575.745.492$ Replies. Mit dem nach Tabelle 3.15 mittleren Volumen einer

ICP-Reply von 76 Byte ergibt sich weiterhin das übertragene Datenvolumen $V_{Out_{\mathrm{ICP}}} =$ 40,8 GByte.

Für die Angabe der Inter-Cache-Kommunikation über ICP innerhalb der Cache-Verbünde ist zunächst die Verteilung der HTTP-Requests nach Toplevel Domains zu berücksichtigen. Da bei Cache-Hits keine Kommunikation innerhalb des Cache-Verbundes erforderlich ist, ist weiterhin die Request-Trefferrate in Abhängigkeit von der jeweiligen Toplevel Domain erforderlich. Tabelle 5.4 zeigt, dass die entsprechenden Request-Trefferraten annähernd konstant über alle nach Tabelle 5.2 betrachteten Toplevel Domains sind.

	Partitionen					
	1	**2**		**3**		
Toplevel Domains	all	com+num	!(com+num)	com	de	!(com+de)
Anteil Requests	100,00%	39,96%	60,04%	37,05%	43,68%	19,26%
Request-Trefferrate	32,74%	32,37%	32,98%	32,14%	34,26%	30,44%

Tabelle 5.5: Request-Trefferraten über verschiedene Toplevel Domains

Für die weitere Berechnung bietet sich die Einführung der Wahrscheinlichkeiten p_{TLD} für einen Requests aus der Toplevel Domain TLD und p_{Hit} für die Request-Trefferrate bzw. $(1-p_{Hit})$ für die Wahrscheinlichkeit eines Cache-Miss an. Mit n_i Cache-Servern in Partition i ergibt sich folgende Gleichung:

$$R_{Among_{\mathrm{ICP}}} = R_{Out_{\mathrm{WWW}}} \cdot \sum_{i=1}^{P} p_{TLD_i} \cdot (1-p_{Hit_i}) \cdot (n_i - 1) \tag{5.2}$$

Die nach dieser Gleichung berechneten Ergebnisse sind in Tabelle 5.6 zusammengefasst.

Partitionen	**Cache-Server**		
	4	**8**	**16**
1	863.618.238 Requests 61,1 GByte	1.439.363.730 Requests 101,9 GByte	4.318.091.190 Requests 305,6 GByte
2	94.242.330 Requests 6,7 GByte	471.211.651 Requests 33,4 GByte	659.696.312 Requests 46,7 GByte
3	–	265.831.487 Requests 18,8 GByte	411.249.750 Requests 29,1 GByte

Tabelle 5.6: ICP-Verkehr innerhalb des übergeordneten Cache-Verbundes

Auf eine Verifizierung der hier vorgestellten Berechnungen mit dem Simulationsmodell wird verzichtet. Stattdessen wird das Simulationsmodell eingesetzt, um die

Lastverteilung zwischen den Cache-Servern des jeweiligen Cache-Verbundes zu prüfen. Abbildung 5.14 zeigt die im 5-minütigen Mittel dargestellte Anzahl HTTP-Responses für jeden Cache-Server in den verschiedenen Cache-Verbünden.

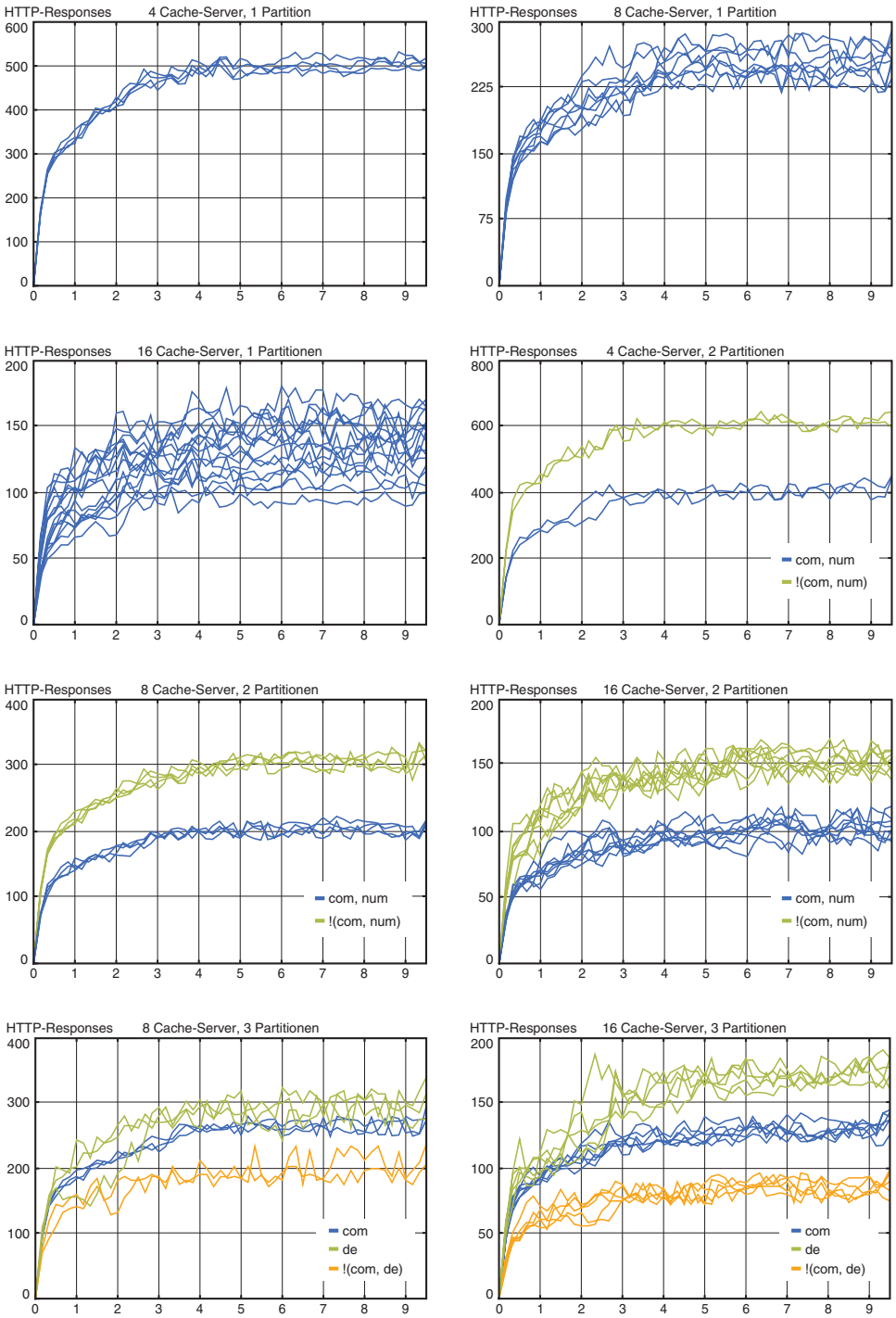


Abbildung 5.14: Verteilung HTTP-Requests in verschiedenen Cache-Verbünden

Da bei der Zuordnung der Toplevel Domains eine Gleichverteilung des übertragenen Datenvolumens zwischen den verschiedenen Partitionen angestrebt wurde, treten bei der Verteilung der HTTP-Requests erhebliche Unterschiede auf. Auf die Ursache hierfür wurde bereits mehrfach in dieser Arbeit hingewiesen (Kapitel 3.3.4.5, Kapitel 4.2.2.1).

Den Darstellungen in Abbildung 5.14 lässt sich entnehmen, dass in Partitionen mit mehreren Cache-Servern eine gleiche Verteilung der HTTP-Requests schwer durchführbar ist. Die untersuchten Cache-Verbünde mit 8 und 16 Cache-Servern in jeweils einer Partition zeigen eine breite Streuung der bearbeiteten HTTP-Responses zwischen den einzelnen Cache-Servern. Obwohl in den Untersuchungen eine günstige Zuordnung der lokalen Cache-Server gewählt wurde, unterscheidet sich in der Partition mit 16 Cache-Servern die Anzahl bearbeiteter HTTP-Responses zwischen zwei Cache-Servern um bis zu 75%. Neben dem hohen Verkehrsaufkommen zur Inter-Cache-Kommunikation liefert die unterschiedliche Last auf den Cache-Servern ein weiteres Argument gegen die Bildung von Cache-Verbünden mit einer hohen Anzahl Cache-Servern in den Partitionen.

Als ein Ergebnis der Untersuchungen lässt sich feststellen, dass eine auf den verfügbaren Protokollen basierende automatisierte optimale Verteilung der Last in einem Cache-Verbund nicht möglich ist. Eine verbesserte Lenkung der HTTP-Requests erfordert die Berücksichtigung der jeweiligen Lastsituation auf den einzelnen Cache-Servern. Entsprechende Komponenten liegen mit den so genannten Load Balancern vor. Load Balancer werden ähnlich zu herkömmlichen Switches vor eine Gruppe von Cache-Servern (Cache-Cluster) geschaltet. Durch Monitoring der Transferzeiten, Datenraten und der Anzahl übertragener HTTP-Responses verfügt der Load Balancer über geeignete Informationen, um die Lastsituation auf den angeschlossenen Cache-Servern beurteilen zu können. Load Balancer können jedoch in der Regel nur in lokalen Netzen unmittelbar vor einem Cache-Cluster eingesetzt werden.

Zusammenfassend lässt sich feststellen, dass mit den formulierten Gleichungen die theoretischen Grundlagen zur eingehenden Untersuchung von Cache-Verbünden vorliegen. Darüber hinaus wurde mit dem Simulationsmodell ein effizientes und flexibles Werkzeug implementiert, mit dem sich auch komplexe Cache-Verbünde hinsichtlich verschiedener Konfigurationen und Lastsituationen untersuchen lassen. Das Modell bietet die Möglichkeit für zahlreiche Erweiterungen. Erforderlich wäre zum Beispiel die Implementierung verschiedener Ankunftsprozesse, um auch die Auswirkungen burstartigen WWW-Verkehrs geeignet untersuchen zu können. Weiterhin könnten neue Protokolle zur Inter-Cache-Kommunikation oder zur Cache-Kohärenz prototypisch an dem Modell erprobt werden.

Kapitel 6

Zusammenfassung und Ausblick

Der wissenschaftliche Schwerpunkt dieser Arbeit liegt in der strukturierten analytischen Untersuchung der Verkehrsströme in einem Cache-Verbund unter Berücksichtigung der übertragenen Datenvolumen und der Übertragungszeiten. Das Vorgehen gründet sich zunächst auf die theoretische Betrachtung der in einem Cache-Verbund auftretenden Kommunikationsbeziehungen, wobei sowohl die Übertragung der Nutzdaten über HTTP als auch die Protokolle zur Inter-Cache-Kommunikation *Inter Cache Protocol*, *Hypertext Cache Protocol* und *Cache Digests* berücksichtigt werden.

Die Ergebnisse dieser Untersuchungen werden durch Messungen aus dem praktischen Betrieb eines Cache-Verbundes im Wissenschaftsnetz verifiziert. Hierbei wird eine Klassifizierung hergeleitet, nach der sich allgemein die in einem Cache-Verbund protokollierten Übertragungsvorgänge einordnen und entsprechend auswerten lassen. Der Vergleich von berechneten und gemessenen Werten bestätigt die formulierten Zusammenhänge. Anhand der nach Sibling- und Parent-Beziehungen korrigierten Betrachtung des ICP-Verkehrs wird beispielhaft verdeutlicht, dass sich diese Gleichungen nur bei genauer Kenntnis der dem Cache-Verbund zugrunde liegenden Struktur und unter Beachtung der vorgeschriebenen Randbedingungen korrekt anwenden lassen.

Aus den praktischen Messungen wird ein Verfahren entwickelt, um die relevanten Datenströme innerhalb des über einen Cache-Verbund übertragenen WWW-Verkehrs zu identifizieren. Der Kern des Verfahrens liegt in der Bestimmung der Parameter, von denen die Datenvolumen der übertragenen Objekte und die Übertragungsraten abhängen. Als mögliche Parameter werden Mimetypes, Toplevel Domains, HTTP-Codes und Cache-Hits bzw. -Misses auf den Cache-Servern betrachtet. In einem weiteren Schritt werden geeignete Grenzwerte untersucht, mit denen eine verlustbehaftete Reduzierung der betrachteten Datenströme unter Beibehaltung der gesamten Verkehrscharakteristik möglich ist. Als Ergebnis wird festgestellt, dass von den theoretisch möglichen 1.764 verschiedenen Kombinationen lediglich 50 bis 70 Datenströme einen relevanten Anteil an dem auftretenden WWW-Verkehr liefern. Der Vergleich der betrachteten Datensätze aus fünf ver-

schiedenen Zeiträumen zeigt weiterhin, dass das Verfahren nicht statisch anwendbar ist, sondern je nach Datensatz verschiedene relevante Datenströme liefert.

Die mathematische Modellierung der identifizierten Datenströme liefert die Datenbasis für die Eingangslast des entwickelten Simulationsmodells. Es ist besonders hervorzuheben, dass die mathematische Modellierung überwiegend über Verteilungsfunktionen mit long-tail Eigenschaften durchgeführt wird. Nicht zuletzt aus diesem Grund scheint eine geschlossene analytische Modellierung von Cache-Verbünden, die in dieser Arbeit nicht weiter betrachtet wird, nur unter hohem Aufwand durchführbar.

Der Implementierung des Simulationsmodells wird eine Betrachtung geeigneter Warteschlangenmodelle für einen Cache-Verbund vorangestellt. Ausgehend von dem Modell werden Algorithmen entwickelt, mit denen sich die Kommunikationsabläufe in einem Cache-Verbund für das Simulationsmodell abbilden lassen. Dabei kann erneut auf die Formulierung der allgemeinen Verkehrsbeziehungen zurückgegriffen werden.

Die durchgeführten Simulationen belegen, dass der entwickelte Lastgenerator eine sehr gute Näherung des realen Verkehrsaufkommens sowohl bezüglich des übertragenen Datenvolumens als auch der Übertragungsdauern liefert. Mit mehreren Simulationsläufen werden abschließend beispielhaft verschiedene Szenarien untersucht. Insbesondere aus der Betrachtung von Störungen ergeben sich Erkenntnisse, die für die Konzeption und den praktischen Betrieb eines Cache-Verbundes von Bedeutung sind und sich ohne den Einsatz von Simulationen nur schwer erschließen.

Ausblick

Entwicklungen, die das dem WWW zugrunde liegende HTTP oder die Inter-Cache-Protokolle maßgeblich verändern oder ergänzen, sind derzeit nicht abzusehen. Die Entwicklung von HTTP wurde mit der Version HTTP/1.1 abgeschlossen. Derzeitige Aktivitäten des W3C beschränken sich auf die Internationalisierung und die verbesserte Darstellung von Inhalten. Aktivitäten, die auf eine Verbesserung der Übertragungsverfahren zielen, sind nicht zu erkennen.

Seitens der IETF wurden zwei Arbeitsgruppen initiiert, die sich mit der Verteilung von Objekten im World Wide Web befassten. Die Arbeitsgruppe *Web Replication and Caching* (WREC) wurde im März 2001 aufgelöst, die Auflösung der noch aktiven Arbeitsgruppe *Web Intermediaries* (WEBI) steht offenbar unmittelbar bevor. Obwohl in beiden Gruppen verschiedene Protokolle für die Verbesserung der Konsistenz in Cache-Verbünden vorgeschlagen und diskutiert wurden, ist lediglich das *Resource Update Protocol* (RUP) [CLDH02] noch von aktueller Bedeutung. Ob es allerdings in den Status eines RFC übergeht und Implementierungen zur Verfügung stehen werden, ist derzeit offen.

Mit der ausschließlichen Zielsetzung, Übertragungszeiten im WWW zu verringern, haben sich neben dem Caching mittlerweile so genannte *Content Distribution Networks* (CDN) etabliert. Ein CDN besteht aus einem Netz von Servern, auf denen Inhalte bestimmter WWW-Server weitgehend vollständig gespiegelt werden. Durch gezielte Manipulation von Routing- oder DNS-Einträgen werden die Klienten nicht auf die originalen WWW-Server, sondern transparent auf die nächsten Server im CDN gelenkt. Hierfür ist zum Teil auch eine Anpassung der in den gespiegelten WWW-Servern enthaltenen Links erforderlich.

Hinter der Installation von CDNs stehen kommerzielle Interessen. Da der Betrieb eines CDN mit hohen Kosten verbunden ist, werden seitens der Betreiber Entgelte von den Content-Anbietern für die Spiegelung ihrer Inhalte verlangt. Daraus folgt, dass ein kommerzielles Interesse der Content-Anbieter an einer verbesserten Verfügbarkeit ihrer Inhalte vorliegen muss. Dieses Interesse kann zumindest bei Einrichtungen aus dem Hochschulbereich nicht vorausgesetzt werden, weshalb hier die Nutzung von CDN ohne Bedeutung ist. Gleichwohl werden im deutschen Wissenschaftsnetz CDN-Server der Firma Akamai betrieben, auf denen zahlreiche WWW-Server gespiegelt sind. Offizielle Statistiken, die belegen, welches Datenvolumen von den CDN-Servern im Wissenschaftsnetz übertragen wurde, wurden bisher nicht veröffentlicht.

Eine weitere Entwicklung im Internet, die das Verkehrsaufkommen und das Caching im WWW stark beeinflussen wird, zeichnet sich mit den so genannten *Peer-to-Peer-Netzen* (P2P-Netze) ab. Ursprung dieser Netze sind Tauschbörsen von Audio- oder Videodateien wie Napster oder Kazaa. P2P-Netze zeichnen sich dadurch aus, dass keine dedizierten Server existieren. Jeder Rechner in einem P2P-Netz arbeitet gleichzeitig als Klient und Server. Der Austausch der Daten zwischen den Rechnern erfolgt über nicht näher spezifizierte Protokolle, die durch Verwendung ständig wechselnder Ports nur schwer innerhalb des gesamten Verkehrs im Internet identifiziert werden können.

Da die Daten, die in P2P-Netzen übertragen werden, im Wesentlichen aus Objekten der Mimetypes Application, Audio und Video bestehen, ist zu vermuten, dass sich im WWW eine Verschiebung zugunsten von Objekten der Mimetypes Image und Text ergibt. Allein aus den in dieser Arbeit dargestellten Unterschieden zwischen den Größenverteilungen verschiedener Mimetypes ist eine deutliche Veränderung des WWW-Verkehrs zu erwarten. Grundlegende Untersuchungen von P2P-Netzen und die Auswirkungen auf die Verkehrscharakteristik im WWW sind bisher jedoch nicht verfügbar. Aufgrund des hohen Datenvolumens, das in P2P-Netzen ausgetauscht wird und das zum Teil das Aufkommen von WWW-Verkehr übersteigt, sind nähere Kenntnisse hierüber aktuell von großem Interesse.

Auch wenn sich neben dem WWW neuartige Dienste zur Übertragung und Verteilung von Inhalten über das Internet bilden, wird die Bedeutung des WWW in den nächsten Jahren weiter zunehmen. Einhergehend wird auch die Nutzung von Cache-

Servern steigen, wobei offen ist, ob die Begründung für deren Einsatz in der Schonung von Kapazitäten auf dem Netz oder in der Verringerung von Zugriffszeiten liegt. Aus Sicht der Netzbetreiber ist festzustellen, dass mittlerweile die Verwendung transparenter Cache-Server, die allein eine hohe und ausfalltolerante Nutzung sicherstellen, ohne Schwierigkeiten möglich ist.

Demgegenüber kann die zukünftige Bedeutung von Cache-Verbünden derzeit nur schwer abgeschätzt werden. So wurde mit dem Ende des in dieser Arbeit dargestellten DFN-Projekts empfohlen, den Betrieb des DFN-Cache-Verbundes einzustellen. Ausschlaggebendes Argument waren die im Vergleich zum gesamten Verkehrsaufkommen im Wissenschaftsnetz geringen Einsparungen sowie die hohen Investitionskosten für eine angemessene Hardware-Plattform. Diese Entscheidung ist jedoch allein auf das Wissenschaftsnetz bezogen und lässt sich nicht unmittelbar auf andere Netze übertragen. So bietet sich der Einsatz von Cache-Verbünden besonders für standortübergreifende Intranets an. Weiterhin ist bekannt, dass der überwiegende Teil hardware-basierter Cache-Server im Access-Bereich großer ISPs eingesetzt wird. In der Regel werden diese Cache-Server unter Verwendung von Inter-Cache-Kommunikation zu Cache-Clustern konfiguriert. Ohne Cache-Server könnten hier, besonders bei fortschreitender Verbreitung von DSL-Technologien, die erforderlichen Kapazitäten in den Hauptverkehrszeiten nicht angemessen bedient werden.

Das Prinzip des verteilten Caching wird im Internet mittlerweile nicht mehr auf statische Inhalte im WWW begrenzt. Aktuelle Veröffentlichungen untersuchen z. B., wie weit sich durch Caching die Qualität der Übertragung von Audio- und Video-Streams steigern lässt [RYHE99] [BGHP00]. Es ist zu erwarten, dass Cache-Server in entsprechenden Anpassungen zukünftig auch zur Unterstützung neuer Dienste im Internet Verwendung finden.

Anhang A

Begleitende Untersuchungen

A.1 Untersuchung des Zeitverhaltens von WWW-Servern

In den Spezifikationen von HTTP wird gefordert, dass die Uhren von WWW-Servern über das *Network Time Protocol* (NTP) [Mil92] mit Zeitreferenzen synchronisiert werden sollen. Um zu prüfen, wieweit diese Forderung erfüllt wird, wurden in einem Experiment die Datumsangaben von 100.000 verschiedenen WWW-Servern ausgewertet. Die Adressen der WWW-Server wurden aus den Logdateien vom 4. bis 8. Dezember 2000 der vier DFN-Cache-Server gewonnen.

Die Übertragung der ausgewählten Objekte von den WWW-Servern erfolgte am 9. und 10. Dezember 2000 mit dem Programm *Pavuk* [Pav01]. Das Programm lief gleichzeitig auf zehn Systemen am RVS mit jeweils bis zu 15 parallelen Prozessen. Auf allen Systemen wurden die Rechneruhren über NTP mit einer hochgenauen lokalen Zeitreferenz (Meinberg GPS 166) synchronisiert. Die maximale Abweichung der Rechneruhren von dieser Zeitreferenz betrug weniger als 100 ms.

Um die Differenz der Uhren (Offset) zwischen einem WWW-Server und einem WWW-Klienten zu ermitteln, muss zunächst eine Referenzzeit auf dem WWW-Klienten definiert werden. Im Rahmen dieses Experiments wird diese Zeit T_{Ref} aus dem arithmetischen Mittel von Beginn T_1 und Ende T_2 der Übertragung berechnet. Die Differenz zwischen T_{Ref} und der Datumsangabe T_{Server} in dem General-Header `Date:` der HTTP-Response ergibt eine Abschätzung für den Offset T_{Offset}. Die Zeitangaben erfolgen mit einer Auflösung von 1 Sekunde.

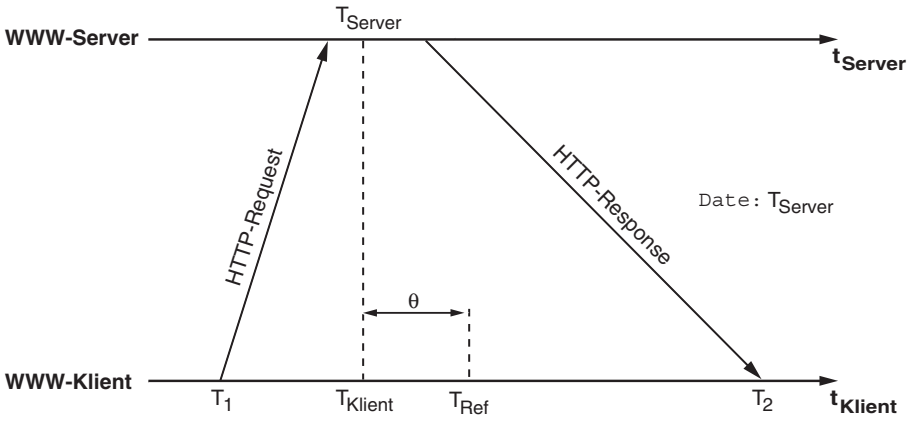


Abbildung A.1: Berechnung Offset zwischen WWW-Server und WWW-Klient

Für eine exakte Berechnung des Offset ist es notwendig, dass das Datum in der HTTP-Response auf dem WWW-Server zu demselben Zeitpunkt wie die Referenzzeit auf dem WWW-Klienten ermittelt wird, d. h. $T_{\text{Ref}} = T_{\text{Klient}}$. Da in der Regel $T_{\text{Ref}} \neq T_{\text{Klient}}$ gilt und T_{Klient} nicht angegeben werden kann, enthält jede der n Berechnungen des Offset einen nicht quantifizierbaren Fehler θ_n. Zusammengefasst ergeben sich folgende Berechnungen:

$$\begin{aligned} &\text{Referenzzeit } T_{\text{Ref}} = \frac{T_1 + T_2}{2} \\ &\text{Offset} \qquad T_{\text{Offset}} = T_{\text{Ref}} - T_{\text{Server}} + \theta \\ &\text{Fehler} \qquad \theta = T_{\text{Ref}} - T_{\text{Klient}} \qquad T_{\text{Klient}} \text{ ist nicht bekannt!} \end{aligned} \tag{A.1}$$

Die Wahrscheinlichkeit, dass große Fehler auftreten, steigt mit hohen Übertragungszeiten. Daher wurde der Timeout bei jeder Übertragung auf 1 Minute begrenzt. Dennoch ist festzuhalten, dass mit diesem Experiment lediglich qualitative Aussagen über die zeitliche Differenz der Uhren auf WWW-Servern und WWW-Klienten abgeleitet werden können.

Aufgrund des geringen Timeouts konnten von den 100.000 Startseiten nur 82.705 erfolgreich abgerufen werden, von denen wiederum 80.199 (97,0%) einen gültigen Header `Date:` enthielten. Die ermittelten Offsets sind in Abbildung A.2 dargestellt. Um die Interpretation zu vereinfachen, sind in das Diagramm zusätzliche Marken eingetragen, an denen der Offset ±2 Minuten und ±10 Minuten beträgt.

Bewertung

Aufgrund des Messverfahrens sind quantitative Aussagen über Offsets im Bereich weniger Sekunden nicht möglich. Es kann jedoch festgehalten werden, dass nur ca.

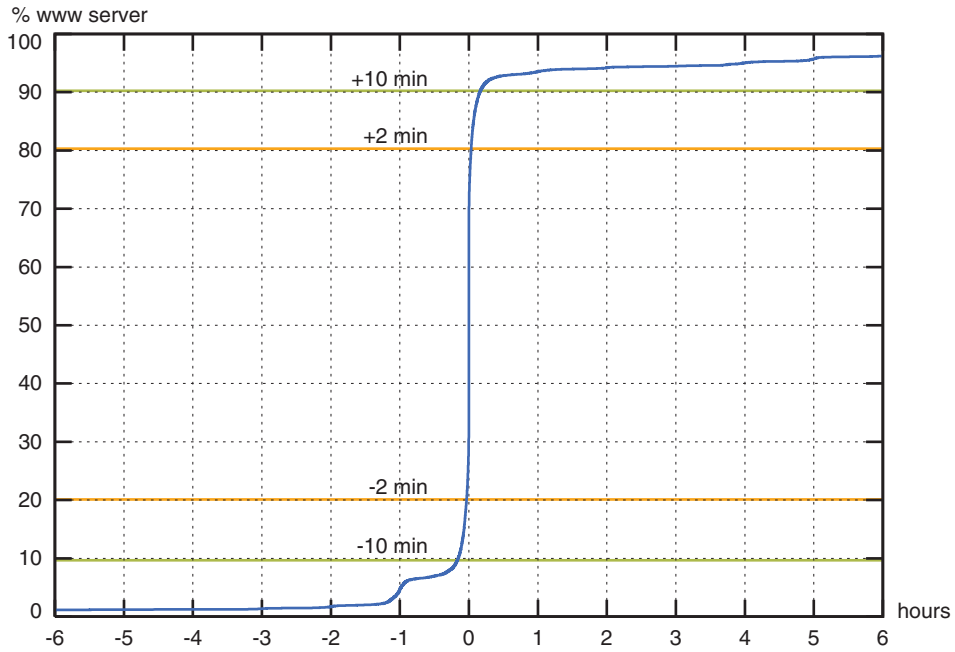


Abbildung A.2: Zeitlicher Offset von WWW-Servern

60% der WWW-Server einen Offset von weniger als 2 Minuten aufweisen. Ein Offset von weniger als 10 Minuten wird von 80% der WWW-Server erreicht. Auffällig ist weiterhin der Absatz bei einem Offset von -1 Stunde. Dieser Offset erklärt sich dadurch, dass die Uhren dieser WWW-Server auf die Mitteleuropäische Zeitzone (MET) und nicht auf die im World Wide Web geforderte Greenwich Mean Time (GMT) konfiguriert sind.

A.2 Untersuchung der Zeitangaben von Objekten im WWW

In einer Untersuchung über die Angaben in den Headern von HTTP-Responses wurden folgende Fragestellungen betrachtet:

- Wie viele Header enthalten Informationen, mit denen ein Cache-Server die Aktualität der Objekte beurteilen kann?
- Wie hoch ist der Anteil an Objekten, die durch entsprechende Angaben in den Headern eine Speicherung auf Cache-Servern verbieten oder eine ständige Aktualisierung erzwingen?
- Welche Auswirkungen haben die Refresh Pattern auf die Häufigkeit der Aktualisierungen?

Für die Beantwortung der Fragen wurden die Header-Informationen von 750.000 Objekten aus dem World Wide Web ausgewertet. Die URLs der Objekte wurden aus den Logdateien der DFN-Cache-Server vom 7. – 8. Dezember 2000 gewonnen. Die Auswahl umfasste nur solche URLs, die nach den HTTP-Statuscodes 2xx oder 304 in den Logdateien bereits erfolgreich übertragen oder aktualisiert werden konnten. Dabei blieben CGIs unberücksichtigt, da deren URLs in den Logdateien nur unvollständig aufgezeichnet werden.

Die Übertragung der ausgewählten Objekte von den WWW-Servern erfolgte am 9. und 10. Dezember 2000 mit dem Programm *Pavuk* [Pav01]. Das Programm lief gleichzeitig auf zehn Systemen am RVS mit jeweils bis zu 15 parallelen Prozessen. Auf allen Systemen wurden die Rechneruhren über NTP mit einer lokalen Zeitreferenz synchronisiert. Von den 750.000 Objekten konnten lediglich 5.502 nicht erfolgreich abgerufen werden. Häufigste Ursachen für die Fehler waren Timeouts beim Verbindungausbau zum WWW-Server oder bei der Übertragung.

In Tabelle A.1 sind die Ergebnisse aus dem ersten Teil der Untersuchung dargestellt. Mit 97,3% enthält ein hoher Anteil der Objekte einen General-Header `Date:`. Das Alter eines Objekts zum Zeitpunkt der Auslieferung vom WWW-Server wird durch die Differenz aus dem General-Header `Date:` und dem Entity-Header `Last-Modified:` berechnet. Unter der Nebenbedingung, dass die Differenz positiv sein muss, lässt sich das Alter für 86,1% der Objekte angeben. Ein Entity-Header `Expires:` findet sich nur in 3,7% der Objekte.

Objekte gesamt	Date:	Last-Modified:	Date: > Last-Modified:	Expires:
744.498 100,0%	724.659 97,3%	648.475 87,1%	641.279 86,1%	28.102 3,8%

Tabelle A.1: Nutzung von Datumsangaben in HTTP-Responses

Die Speicherung von Objekten auf einem Cache-Server kann seit HTTP/1.1 durch den General-Header `Cache-Control:` gesteuert werden. Die Auswertung der ent-

sprechenden Header in den untersuchten HTTP-Responses ist in Tabelle A.2 dargestellt. Insgesamt enthalten 5,5% der Objekte einen solchen Header. Die Verteilung der einzelnen Direktiven ist der Tabelle zu entnehmen. Dabei ist zu berücksichtigen, dass in einem Header mehrere Direktiven gleichzeitig angegeben werden können.

Objekte gesamt	**Cache-Control:**					
	gesamt	**private**	**no-cache**	**no-store**	**max-age**	**unknown**
744.498	40.601 5,5%	21.647 2,9%	22.442 3%	99 0,0%	9.836 1,3%	2.417 0,3%

Tabelle A.2: Nutzung von `Cache-Control:` in HTTP-Responses

In einem nächsten Schritt wurde untersucht, wie die Header `Expires:` und `Cache-Control: max-age=n` genutzt werden. In Abbildung A.3 ist zu erkennen, dass `Expires:` in ca. 70% und `Cache-Control: max-age=n` in ca. 45% der Fälle verwendet wird, um eine langfristige Speicherung der Objekte ohne Aktualisierung zu verhindern. Allerdings gestattet die Angabe von `Expires:` in 15% und von `Cache-Control: max-age=n` in 30% der Fälle eine Speicherung der Objekte für die Dauer von mehr als einer Woche.

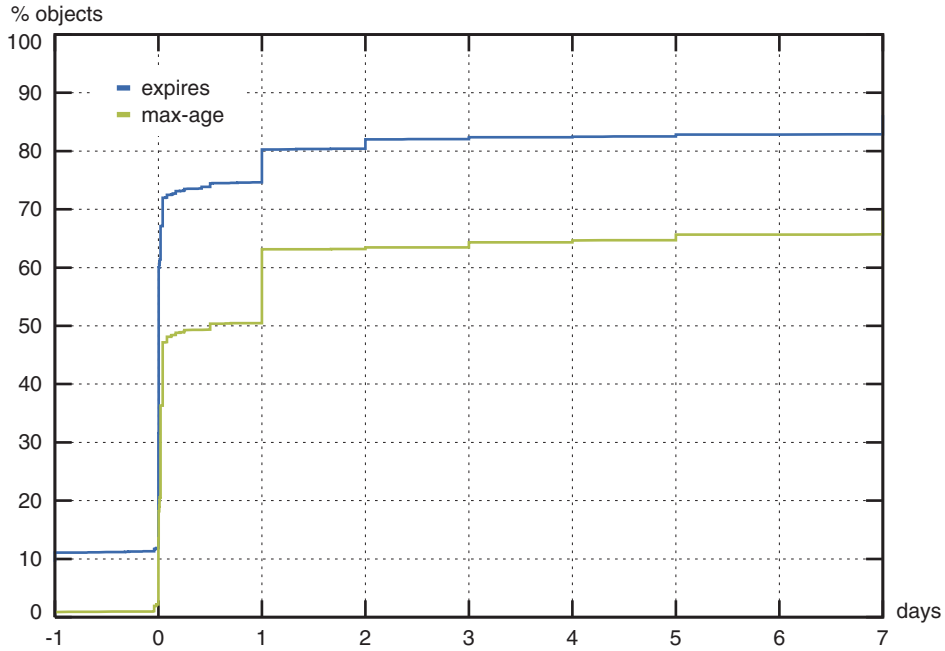


Abbildung A.3: Auswirkung `Expires:` und `Cache-Control: max-age=n`

Der Einfluss der Angaben in den Headern der HTTP-Responses auf die Speicherung und Aktualisierung der Objekte auf einem Cache-Server ist in Tabelle A.3 zusammengefasst. Es lässt sich feststellen, dass 85,7% der Objekte auf Cache-Servern gespeichert und für einen bestimmten Zeitraum ohne vorhergehende Aktualisierung

ausgeliefert werden können. Welche Dauer dieser Zeitraum umfasst, wird in den nachfolgenden Untersuchungen betrachtet.

Objekte gesamt	Speicherung untersagt	ständige Aktualisierung		Caching möglich
	`private` oder `no-store`	`no-cache` oder `Expires: <= Date:` oder `max-age=n <= 0`	`Date: < Last-Modified:`	
744.498	23.698 3,2%	9.728 1,3%	72.720 9,8%	638.352 85,7%

Tabelle A.3: Auswirkung von Angaben in HTTP-Header auf Caching

In einem zweiten Schritt wird untersucht, wieweit die Refresh Pattern die Häufigkeit der Aktualisierungen beeinflussen. In Abbildung A.4 ist dargestellt, über welchen Zeitraum die untersuchten Objekte ohne vorhergehende Aktualisierung vom Cache-Server ausgeliefert werden können. Die Berechnungen orientieren sich an dem Algorithmus der Caching Software Squid (Abbildung 2.10). Als Parameter `CONF_MIN` wird der Standardwert 0 vorausgesetzt. Der Parameter `CONF_PERCENT` ist neben dem Standardwert 0.2 für drei weitere Werte 0.1, 0.05 und 0.01 aufgetragen.

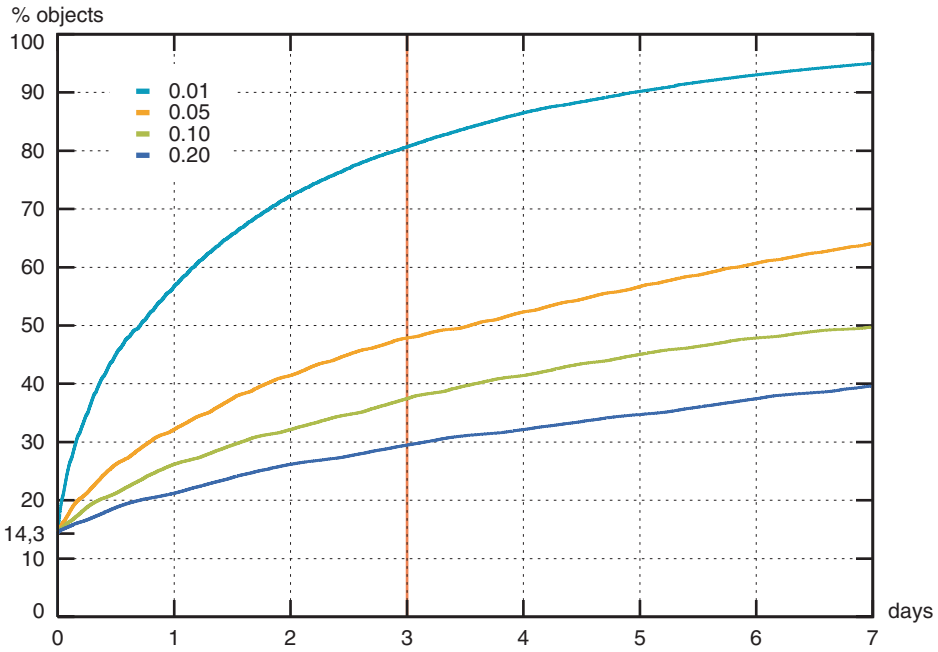


Abbildung A.4: Auswirkung Parameter für Refresh Pattern

Bei den Berechnungen wurde der Parameter `CONF_MAX` nicht berücksichtigt. Dieser Parameter bestimmt die maximale Verweilzeit, nach der spätestens eine Aktualisierung des Objekts notwendig ist. In dem Diagramm lässt sich der Parameter durch

eine senkrechte Linie zu dem entsprechenden Zeitpunkt darstellen. Als Beispiel ist für CONF_MAX der Standardwert von drei Tagen eingetragen.

Es ist zu beachten, dass der Parameter CONF_PERCENT mit dem Standardwert von 0.2 nur ca. 15% der Objekte beeinflusst. Ein Hinweis auf die geringe Auswirkung dieses Parameters ergibt sich aus dem Alter der Objekte zum Zeitpunkt der Übertragung. In Abbildung A.5 wird das Alter aller untersuchten Objekte dargestellt. In dem Diagramm wird ein Alter von bis zu zwei Jahren betrachtet.

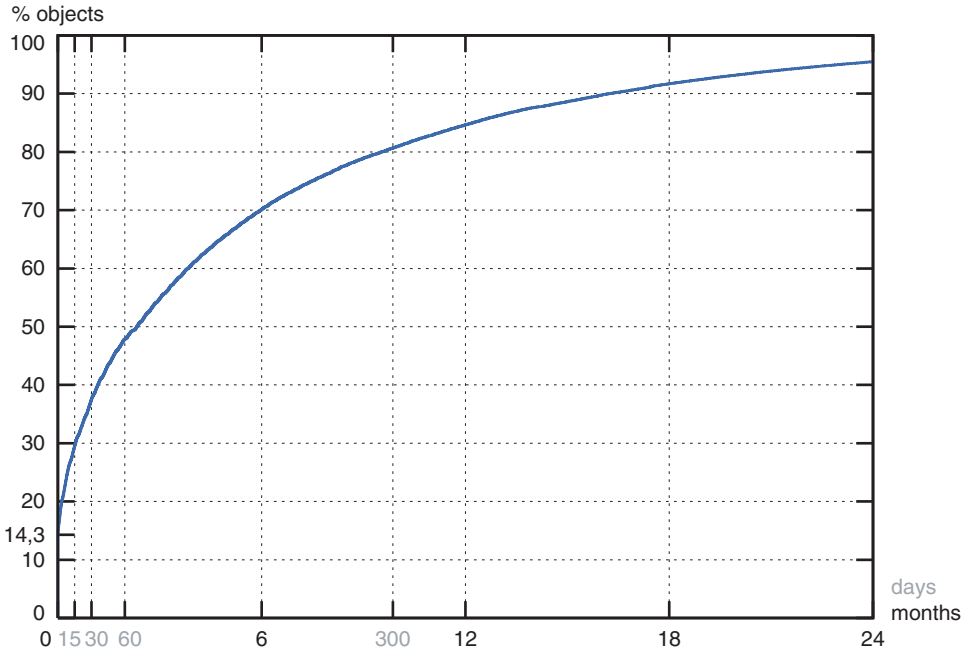


Abbildung A.5: Alter von Objekten zum Zeitpunkt der Übertragung

Aus Abbildung A.5 kann abgelesen werden, zu welchen Anteilen die Parameter CONF_PERCENT und CONF_MAX die Häufigkeit der Aktualisierungen beeinflussen. Hierfür muss zunächst der Quotient CONF_MAX/CONF_PERCENT gebildet werden. Der Quotient gibt das Alter LM_AGE eines Objekts an, dessen Aktualität noch mit dem Faktor CONF_PERCENT beurteilt wird, bevor es die maximale Verweildauer CONF_MAX überschreitet. Aus den Standardwerten ergibt sich ein Alter LM_AGE von 15 Tagen.

Die Altersverteilung in Abbildung A.5 zeigt, dass ca. 29,5% der Objekte das Alter von 15 Tagen unterschreiten. Abzüglich der 14,3% Objekte, die stets aktualisiert werden müssen (s. Tabelle A.3), ergibt sich ein Anteil von 15,2% Objekten, deren Aktualität nach dem Parameter CONF_PERCENT beurteilt wird. Dieser Anteil wurde bereits anhand Abbildung A.4 festgestellt.

Entsprechende Betrachtungen können auch für die Werte 0.1, 0.05 und 0.01 des Parameters CONF_PERCENT durchgeführt werden. Die resultierenden Altersangaben

von 30, 60 und 300 Tagen sind ebenfalls in Abbildung A.5 eingetragen. Die Schnittpunkte der Altersangaben mit der Altersverteilung ergeben die Prozentwerte, die in Abbildung A.4 zum Zeitpunkt „3 days“ abgelesen werden können.

Zusammenfassung

Nur 3,2% der untersuchten Objekte enthalten Header-Informationen, die die Speicherung auf Cache-Servern verhindern. Weitere 11,1% enthalten Angaben, die zwar das Speichern auf Cache-Servern ermöglichen, jedoch eine ständige Aktualisierung erfordern. Demgegenüber können 70% der Objekte während der maximalen Verweildauer von drei Tagen ohne Aktualisierung ausgeliefert werden. Die Aktualität der restlichen ca. 15% wird anhand der aktuellen Verweildauer und des Alters zum Zeitpunkt der Übertragung geschätzt.

Die genannten Angaben beziehen sich auf die Standardwerte der Refresh Pattern. Hierbei beeinflusst der Parameter `CONF_MAX` maßgeblich die Häufigkeit der Aktualisierung von Objekten in einem Cache-Server. Dem Faktor `CONF_PERCENT` kommt nur eine geringe Bedeutung zu, was durch die Altersverteilung der Objekte begründet ist. Erst bei wesentlich kleineren Werten als 0.2 steigt die Bedeutung dieses Faktors an.

Bei den Untersuchungen ist zu berücksichtigen, dass die URLs aus den Logdateien übergeordneter Cache-Server gewonnen wurden. Objekte, die längere Zeit in Caches vorgehalten werden können, werden bereits von Browser-Caches oder lokalen Cache-Servern gespeichert. Daher ist davon auszugehen, dass der Anteil an Objekten, die nicht oder nur kurzzeitig gespeichert werden können, auf übergeordneten Cache-Servern höher ist als auf lokalen Cache-Servern oder in Browser-Caches. Daraus folgt, dass der tatsächliche Anteil an Objekten, die sich nicht speichern lassen oder ständig aktualisiert werden müssen, im World Wide Web geringer sein könnte als in Tabelle A.3 dargestellt.

A.3 Untersuchung von HTCP-Nachrichten

Während der Durchführung des DFN-Cache-Projektes stand noch keine stabile Implementierung von HTCP in Squid zur Verfügung. Folglich konnte HTCP auf den DFN-Caches nicht als Protokoll für die Inter-Cache-Kommunikation erprobt werden. Die Ermittlung der für die Modellierung von HTCP notwendigen charakteristischen Merkmale erfolgte daher in einem lokalen Experiment. Über einen Zeitraum von 7 Tagen, vom 3.–9. September 2001, wurden dem lokalen Cache-Server des RRZN zwei übergeordnete Neighbors im lokalen Netz zugewiesen. Die Inter-Cache-Kommunikation zwischen dem lokalen Cache-Server und den beiden Neighbors erfolgte ausschließlich über HTCP.

Während des betrachteten Zeitraums empfing der lokale Cache-Server 3.1050.148 HTTP-Requests von WWW-Klienten, wovon 23% direkt beantwortet werden konnten. Der Abruf der restlichen 2.390.861 HTTP-Requests erfolgte über die beiden Neighbors. Da jedem Abruf ein HTCP-Request an beide Neighbors mit entsprechenden HTCP-Responses vorausging, konnten im Rahmen des Experiments 9.520.413 HTCP-Nachrichten aufgezeichnet und ausgewertet werden.

Auf den beteiligten Cache-Servern wurde die Software Squid in der Version 2.4.STABLE2 eingesetzt. Es zeigte sich, dass die Implementierung von HTCP in dieser Version zwar stabil, jedoch noch nicht vollständig ist. Daher unterliegen die Untersuchungen folgenden Einschränkungen:

- Squid speichert keine Meta-Informationen der Objekte im Hauptspeicher. Das bedeutet, das auch bei einem Cache-Hit keine Zeitangaben in die HTCP-Response eingetragen werden. Ausnahmen bilden die Objekte, die sich zum Zeitpunkt des Cache-Hits noch im Hauptspeicher befinden.
- Jede HTCP-Response enthält die Textfelder `AGE`, `LAST-MODIFIED` und `EXPIRES`. Bei einem Cache-Miss enthält jedes der Felder einen Eintrag der Länge 0 Byte.
- Squid unterstützt noch keine Authentifizierung von HTCP, so dass die untersuchten HTCP-Nachrichten lediglich aus den Elementen `HEADER` und `DATA` bestehen (s. Abbildung A.6).

Die Ermittlung der Größenverteilung von HTCP-Nachrichten ergibt sich im Folgenden aus der Auswertung der übertragenen HTCP-Nachrichten sowie einer Reihe von Annahmen. Da für die Modellierung eine vollständige Implementierung von HTCP angenommen wird, werden sämtliche Elemente von HTCP-Nachrichten betrachtet. Weiterhin wird vorausgesetzt, dass Zeitinformationen der Objekte bei jedem Cache-Hit zur Verfügung stehen.

In Abbildung A.6 sind die vollständigen Formate von HTCP-Nachrichten, die von der untersuchten Implementierung in Squid generiert werden, zuzüglich des Nachrichtenelements `AUTH` dargestellt.

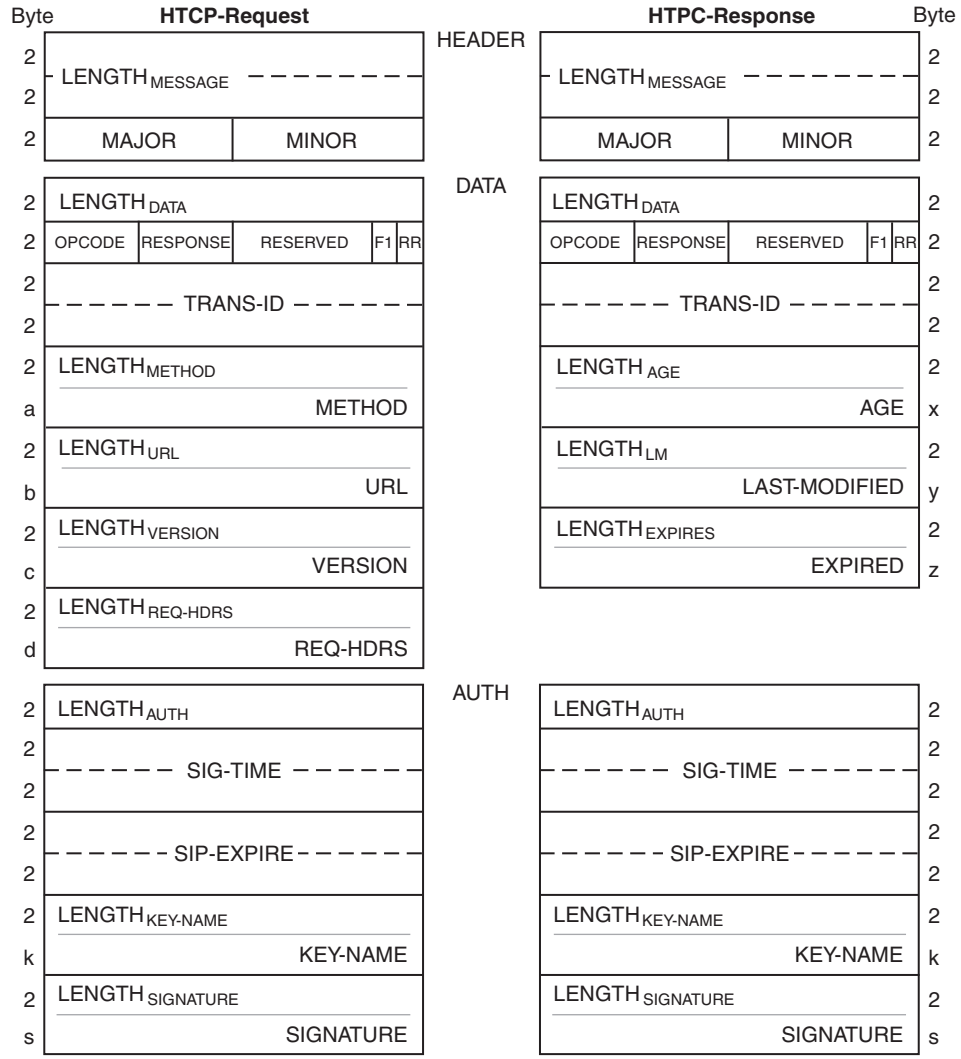


Abbildung A.6: Aufbau von HTCP-Nachrichten in Squid 2.4STABLE2

Die Größe des Elements AUTH einer HTCP-Nachricht kann aus der Darstellung abgeleitet werden. Der Name KEY-NAME signalisiert dem Empfänger der HTCP-Nachricht, welcher der zuvor ausgetauschten geheimen Schlüssel für die Authentifizierung der Nachricht verwendet werden soll. Es ist davon auszugehen, dass auch in größeren Umgebungen keine hohe Zahl von Schlüsseln auf einem Cache-Server zu verwalten ist, so dass eine Länge der Namen von 4 Bytes ausreichend ist. Der für die Authentifizierung vorgeschlagene Schlüssel nach HMAC-MD5 umfasst 16 Bytes. Somit ergibt sich für die gesamte Länge des Elements AUTH der feste Wert von 34 Bytes in jeder HTCP-Nachricht.

Die Länge des Elements DATA einer HTCP-Nachricht muss für HTCP-Requests und HTCP-Responses unterschiedlich angegeben werden. Das Element DATA in einem HTCP-Request besteht unter anderem aus den vier Textfeldern METHOD, URL, VER-

SION sowie REQ-HDR. Das Feld METHOD enthält in der Regel den Text GET, das Feld VERSION enthält den Eintrag der verwendeten HTTP-Version 0.9, 1.0 oder 1.1. Da jedes Textfeld variabler Länge in HTCP mit einem Wagenrücklauf (*carriage return*, <CR>) sowie dem Byte 0 als Kennzeichner für das Ende einer Zeichenkette abgeschlossen wird, sind die Felder METHOD und VERSION mit einer konstanten Länge von 5 Bytes anzunehmen.

Für die Länge der Felder URL und REQ-HDR lassen sich keine festen Werte angeben. Die Längenverteilung von URLs kann der Untersuchung von ICP-Nachrichten entnommen werden (s. Kapitel 3.2.5.1). Die Länge der in einem HTTP-Request übertragenen Header hängt von der Implementierung der verwendeten WWW-Browser ab. Weiterhin werden die Header Via: aufgenommen (s. Kapitel 2.3.1), so dass die Länge des Feldes REQ-HDR in höheren Ebenen einer Cache-Hierarchie steigt. In Abbildung A.7 sind die Größenverteilungen von HTCP-Requests und den darin enthaltenen HTTP-Headern aufgetragen. Beide Verteilungsfunktionen zeigen einen ähnlichen Verlauf, die Verschiebung zwischen den Kurven ergibt sich aus der Länge der URLs. Die mittlere Größe von HTCP-Requests beträgt 598 Byte.

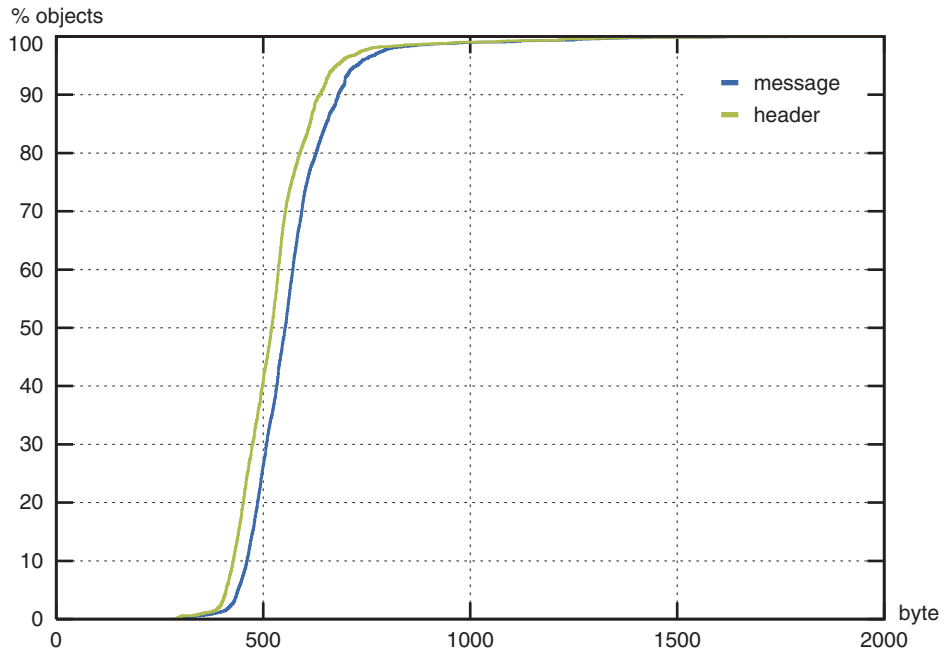


Abbildung A.7: Größe von HTCP-Requests

Im Gegensatz zu ICP enthalten HTCP-Responses keine URLs. Die Zuordnung von HTCP-Responses zu HTCP-Replies auf den lokalen Cache-Servern wird ausschließlich anhand des Feldes TRANS-ID vorgenommen. HTCP-Responses enthalten bei der Signalisierung eines Cache-Hits jedoch die Zeitinformationen des geforderten Objekts. Daraus folgt, dass HTCP-Responses in einem weiteren Schritt für Cache-Hits und Cache-Misses getrennt betrachtet werden müssen.

Für einen Cache-Miss bleiben die Felder AGE, LAST-MODIFIED sowie EXPIRES leer (x=0, y=0, z=0), wodurch die Länge einer entsprechenden HTCP-Response stets 54 Bytes beträgt. Im Falle eines Cache-Hits wird das Feld AGE stets mit der bisherigen Verweildauer des Objekts im Cache-Server gefüllt. Die Felder LAST-MODIFIED und EXPIRES werden dann gefüllt, wenn entsprechende Angaben in der HTTP-Response des WWW-Servers vorhanden waren. Abbildung A.8 zeigt die Größenverteilung der in HTCP-Nachrichten signalisierten Cache-Hits. Die Ursache dieser Sprungstellen liegt in der variierenden Länge des Feldes AGE sowie in den unterschiedlichen Häufigkeiten, mit denen Zeitinformationen für die Objekte angegeben werden können.

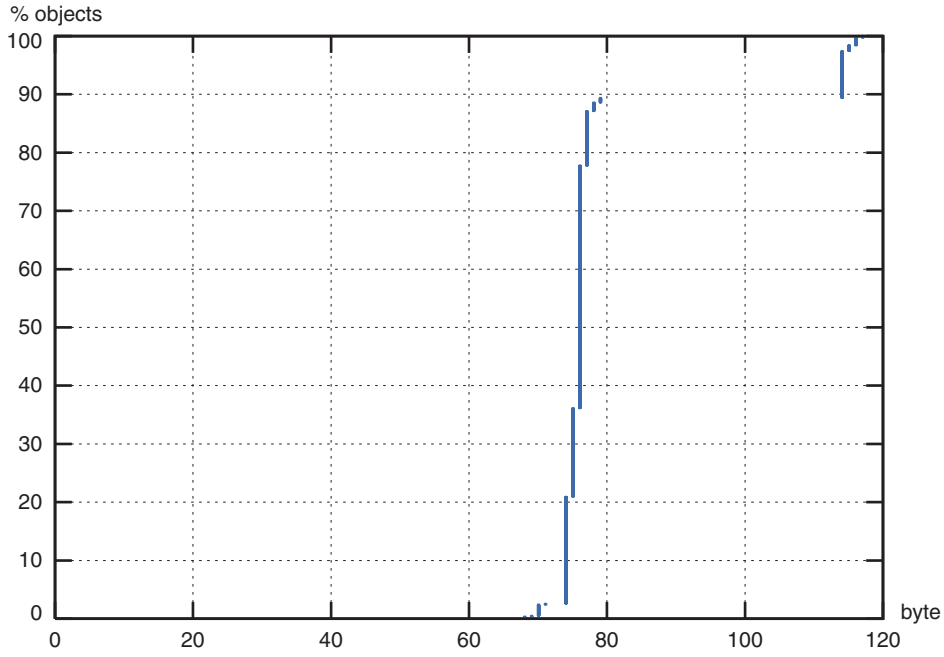


Abbildung A.8: Größe von HTCP-Responses für Cache-Hits

Die Länge des Zahlenwertes in dem Feld AGE variiert in den untersuchten HTCP-Responses in dem Bereich 1 – 7. Für eine Modellierung genügt der Mittelwert von 3.88 bzw. der Median von 4 Bytes. Daraus ergibt sich eine mittlere Länge des Feldes AGE von 11 Bytes. Die Länge des Feldes LAST-MODIFIED beträgt konstant 46 Bytes, die Länge des Feldes EXPIRES 40 Bytes. Die Häufigkeiten, mit denen diese Angaben in den untersuchten HTCP-Requests auftreten, ist zusammen mit der resultierenden Länge der entsprechenden HTCP-Nachrichten in Tabelle A.4 dargestellt.

Header	Anteil	Länge der HTCP-Header [Byte]	Länge HTCP-Nachricht [Byte]
AGE + LAST-MODIFIED	86,8%	11 + 46	105
AGE + EXPIRES	2,5%	11 + 40	99
AGE + LAST-MODIFIED + EXPIRES	10,7%	11 + 46 + 40	151

Tabelle A.4: Häufigkeit von Zeitangaben in HTCP-Responses für Cache-Hits

Hiermit sind die Größen von HTCP-Requests sowie von Cache-Hits und Cache-Misses in HTCP-Responses angegeben. Die Größe eines HTCP-Requests muss anhand der Verteilungsfunktion ermittelt werden. Bei einer HTCP-Response muss zunächst zwischen Cache-Hit und Cache-Miss unterschieden werden. Ein Cache-Miss erzeugt stets eine HTCP-Response von 54 Bytes. Ein Cache-Hit erzeugt in Abhängigkeit von den Header-Informationen eine HTCP-Response von 99, 105 oder 151 Bytes. Zusammen mit den in Tabelle A.4 aufgeführten Häufigkeiten ergibt sich für HTCP-Responses eine mittlere Größe von 110 Byte.

Anhang B

Verteilungsfunktionen

B.1 Exponentialverteilung

Parameter	• Skalierungsparameter $\lambda > 0$ • Lageparameter $-\infty < k < \infty$
PDF	$f_X(x) = P(X = x) = \begin{cases} \lambda e^{-\lambda(x-k)} & x > k,\ \lambda > 0 \\ 0 & x \le k \end{cases}$ (B.1)
CDF	$F_X(x) = P(X \ge x) = \begin{cases} 1 - e^{-\lambda(x-k)} & x > k,\ \lambda > 0 \\ 0 & x \le k \end{cases}$ (B.2)
Mittelwert	$\mu_x = \frac{1}{\lambda} + k$ (B.3)
Varianz	$\sigma_x^2 = \frac{1}{\lambda^2}$ (B.4)
Median	$\tilde{x}_{0,5} = \frac{\ln(2)}{\lambda} + k$ (B.5)
Modalwert	$\tilde{x}_m = k$ (B.6)
Linearisierung	$x' = x \quad y' = \ln\left(\frac{1}{1 - F_X(x)}\right) \quad \Rightarrow \quad \lambda = m \quad b \equiv 0$ (B.7)
Inverse	$1 - e^{-\lambda x} = u \quad \Rightarrow \quad x = -\frac{\ln v}{\lambda} \quad \text{mit} \quad v = 1 - u$ (B.8)
Anmerkung	–

B.2 Normalverteilung

Parameter	• Lageparameter $-\infty < \mu < \infty$ • Skalierungsparameter $\sigma > 0$
PDF	$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{-(x-\mu)^2/(2\sigma^2)} \qquad -\infty \le x \le \infty$ (B.9)
CDF	$F_X(x) = \frac{1}{\sqrt{2\pi}\sigma} \cdot \int_{-\infty}^{x} e^{-(\xi-\mu)^2/(2\sigma^2)} d\xi \qquad -\infty \le x \le \infty$ (B.10)
Mittelwert	$\mu_x = \mu$ (B.11)
Varianz	$\sigma_x^2 = \sigma^2$ (B.12)
Median	$\tilde{x}_{0,5} = \mu$ (B.13)
Modalwert	$\tilde{x}_m = \mu$ (B.14)
MLE	$\hat{\mu} = \overline{X}(n) \qquad \hat{\sigma}^2 = \frac{n-1}{n} \cdot S^2(n)$ (B.15)
Inverse	• Generierung z. B. mit Box-Mueller-Methode $x \sim N(0,1)$ und anschließender Transformation $x' = \mu + \sigma x$.
Anmerkung	• Standardnormalverteilung: $\mu = 0,\ \sigma^2 = 1$

B.3 Logarithmische Normalverteilung

Parameter	• Skalierungsparameter $-\infty < \mu_L < \infty$ • Formparameter $\sigma_L > 0$
PDF	$f_X(x) = \begin{cases} \frac{1}{\sqrt{2\pi}\sigma_L} \cdot \frac{1}{x} \cdot e^{-(\ln(x)-\mu_L)^2/2\sigma_L^2} & x > 0 \\ 0 & x \le 0 \end{cases}$ (B.16)
CDF	$F_X(x) = \begin{cases} \frac{1}{\sqrt{2\pi}\sigma_L} \cdot \int_{-\infty}^{x} \frac{1}{\xi} \cdot e^{-(\ln(\xi)-\mu_L)^2/2\sigma_L^2} d\xi & x > 0 \\ 0 & x \le 0 \end{cases}$ (B.17)
Mittelwert	$\mu_x = e^{(\mu_L + \sigma_L^2/2)}$ (B.18)
Varianz	$\sigma^2_x = e^{(2\mu_L + \sigma_L^2)} \cdot (e^{\sigma_L^2} - 1)$ (B.19)
Median	$\tilde{x}_{0,5} = e^{\mu_L}$ (B.20)
Modalwert	$\tilde{x}_m = e^{\mu_L - \sigma_L^2}$ (B.21)
MLE	$\hat{\mu} = \frac{\sum_{i=1}^{n} \ln X_i}{n} \qquad \hat{\sigma}^2 = \frac{\sum_{i=1}^{n} (\ln X_i - \hat{\mu})^2}{n}$ (B.22)
Inverse	• Generierung z. B. mit Box-Mueller-Methode $x \sim N(0,1)$ und anschließender Transformation $x' = e^{\mu + \sigma x}$.
Anmerkung	• Die logarithmierte Normalverteilung oder Lognormalverteilung entsteht aus der Normalverteilung durch die Transformation der Zufallsvariablen $Y = \ln(X)$. • Lognormalverteilung weist long-tail Eigenschaften auf.

B.4 Paretoverteilung

Parameter	• Formparameter $\alpha > 0$ • Lageparameter $k > 0$
PDF	$f_X(x) = P(X = x) = \begin{cases} \frac{\alpha k^\alpha}{x^{\alpha+1}} & x \geq k \\ 0 & x < k \end{cases}$ (B.23)
CDF	$F_X(x) = \begin{cases} 1 - \left(\frac{k}{x}\right)^\alpha & x \geq k \\ 0 & x < k \end{cases}$ (B.24)
Mittelwert	$\mu_x = \begin{cases} \frac{k\alpha}{\alpha - 1} & \alpha \leq 1 \\ \infty & \alpha > 1 \end{cases}$ (B.25)
Varianz	$\sigma_x^2 = \begin{cases} \frac{k^2 \alpha}{(\alpha-2)(\alpha-1)^2} & \alpha \leq 2 \\ \infty & \alpha > 2 \end{cases}$ (B.26)
Median	$\tilde{x}_{0,5} = k \cdot 2^{1/\alpha}$ (B.27)
Modalwert	$\tilde{x}_m = k$ (B.28)
Linearisierung	$x' = \ln x \quad y' = \ln(1 - F_X(x)) \quad \Rightarrow \quad \alpha = -m \quad k = e^{-b/m}$ (B.29)
Inverse	$1 - \left(\frac{k}{x}\right)^\alpha = u \quad \Rightarrow \quad x = k v^{-1/\alpha} \quad \text{mit } v = 1 - u$ (B.30)
Anmerkung	• Pareto-Verteilung weist power-tail Eigenschaften auf.

B.5 Weibullverteilung

Parameter	• Skalierungsparameter $\alpha > 0$ • Formparameter $\beta > 0$
PDF	$f_X(x) = \begin{cases} \frac{\beta}{\alpha} \cdot \left(\frac{x}{\alpha}\right)^{\beta-1} \cdot e^{-(x/\alpha)^{\beta}} & x > 0 \\ 0 & x < 0 \end{cases}$ (B.31)
CDF	$F_X(x) = \begin{cases} 1 - e^{-(x/\alpha)^{\beta}} & x > 0 \\ 0 & x < 0 \end{cases}$ (B.32)
Mittelwert	$\mu_x = \frac{\alpha}{\beta}\Gamma(\frac{1}{\beta})$ (B.33)
Varianz	$\sigma_x^2 = \frac{\alpha^2}{\beta^2}\left(2\beta\Gamma(\frac{2}{\beta}) - \left(\Gamma(\frac{1}{\beta})\right)^2\right)$ (B.34)
Median	$\tilde{x}_{0,5} = \alpha(\ln(2))^{1/\beta}$ (B.35)
Modalwert	$\tilde{x}_m = \begin{cases} \alpha\left(\frac{\beta-1}{\beta}\right)^{1/\beta} & \beta \geq 1 \\ 0 & 0 < \beta < 1 \end{cases}$ (B.36)
Linearisierung	$x' = \ln x \quad y' = \ln\left(\ln\left(\frac{1}{1-F_X(x)}\right)\right) \Rightarrow \beta = m \quad \alpha = e^{-b/m}$ (B.37)
Inverse	$1 - e^{-(x/\alpha)^{\beta}} = u \quad \Rightarrow \quad x = \alpha(-\ln v)^{1/\beta} \quad \text{mit} \quad v = 1 - u$ (B.38)
Anmerkung	• Weibull-Verteilung weist long-tail Eigenschaften für $\beta < 1$ auf.

B.6 Poissonverteilung

Parameter	• Lageparameter $\lambda > 0$
PDF	$f_X(x) = \mathrm{P}(X = x) = e^{-\lambda}\frac{\lambda^x}{x!} \qquad \lambda > 0, \quad x = 0,1,..,\infty$ (B.39)
CDF	$F_X(x) = \mathrm{P}(X \geq x) = e^{-\lambda}\sum_{k=0}^{x}\frac{\lambda^i}{i!} \qquad \lambda > 0, \quad x = 0,1,..,\infty$ (B.40)
Mittelwert	$\mu_x = \lambda$ (B.41)
Varianz	$\sigma_x^2 = \lambda^2$ (B.42)
Median	– (B.43)
Modalwert	$\tilde{x}_{m_1} = \lambda - 1$ und $\tilde{x}_{m_2} = \lambda$ (B.44)
Linearisierung	–
Inverse	–

Anhang C

Modellierung der Eingangslast

C.1 Datensatz 97: 1. Februar 1997 – 28. Februar 1997

Zwischenankunftszeiten in mittlerer Hauptverkehrsstunde

DFN-Cache-Server	HTTP-Responses in HvStd über 20 Tage	mittlere Ankunftsrate [HTTP-Requests/s]
berlin	138.531	1,924
frankfurt	212.625	2,953
hamburg	94.228	1,309
hannover	197.984	2,750
karlsruhe	184.340	2,560
koeln	281.389	3,908
leipzig	194.578	2,702
muenchen	211.204	2,933
nuernberg	153.245	2,128
stuttgart	203.657	2,829
Summe	**1.871.782**	**25,997**

Verteilung Toplevel Domains

	com	de	net	num	edu	org	other
HTTP-Requests	45,95	20,88	7,75	4,93	3,18	1,77	15,54

Verteilung Mimetypes über Toplevel Domains

	com	de	net	num	edu	org	other
Application	0,93	1,15	0,97	1,96	1,94	2,04	1,13
Audio	0,29	0,14	0,34	0,17	0,41	0,19	0,29
Image	71,48	62,13	66,76	58,98	65,08	58,59	70,47
Text	26,41	35,55	31,52	38,17	31,91	38,22	27,41
Video	0,14	0,07	0,04	0,19	0,22	0,22	0,11
Other	0,75	0,96	0,38	0,53	0,45	0,74	0,59

Verteilung HTTP-Codes über Toplevel Domains

	com	de	net	num	edu	org	other
200	79,87	75,32	77,49	64,52	97,65	98,22	85,55
302	2,16	1,31	15,56	22,58	0,00	0,00	0,00
304	14,64	20,79	5,76	11,40	0,00	0,00	10,82
404	2,22	1,13	0,00	0,00	0,00	0,00	2,08
500	0,00	0,00	0,00	1,50	0,00	0,00	0,00
503	0,00	0,00	0,00	0,00	0,00	0,00	0,00
504	0,00	0,00	0,00	0,00	0,00	0,00	0,00
other	1,12	1,44	1,19	0,00	2,35	1,78	1,55

Hit / Miss

	Statuscode				HTTP-Responses (in %)						
	C_L-C_1	C_1-S	C_1-C_2	C_2-S	com	de	net	num	edu	org	other
Hit	200	–	–	–	3,75	1,85	0,60	0,31	0,15	0,11	0,99
	304	–	–	–	1,20	0,61	0,00	0,00	0,00	0,00	0,20
Miss	200	200	–	–	11,37	3,56	0,76	2,74	0,76	0,36	2,59
	200	304	–	–	0,27	0,00	0,00	0,00	0,00	0,00	0,00
	304	200	–	–	2,03	1,41	0,00	0,54	0,00	0,00	0,27
	304	304	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Neighbor-Hit	200	–	200	–	4,29	3,06	1,56	0,00	0,28	0,25	2,45
	200	–	304	–	0,26	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	200	–	1,48	1,30	0,43	0,00	0,00	0,00	0,77
	304	–	304	–	0,35	0,00	0,00	0,00	0,00	0,00	0,00
Neighbor-Miss	200	–	200	200	15,24	6,60	2,83	0,00	1,79	0,94	6,72
	200	–	200	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	200	–	304	200	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	200	–	304	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	200	200	1,38	0,84	0,00	0,00	0,00	0,00	0,36
	304	–	200	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	304	200	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	304	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Hit	302	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	404	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	500	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	503	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	504	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	999	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Miss	302	302	–	–	0,38	0,00	0,21	1,07	0,00	0,00	0,00
	404	404	–	–	0,41	0,00	0,00	0,00	0,00	0,00	0,00
	500	500	–	–	0,00	0,00	0,00	0,07	0,00	0,00	0,00
	503	503	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	504	504	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	999	999	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Neighbor-Hit	302	–	302	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	404	–	404	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	500	–	500	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	503	–	503	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	504	–	504	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	999	–	999	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Neighbor-Miss	302	–	302	302	0,57	0,26	0,94	0,00	0,00	0,00	0,00
	404	–	404	404	0,57	0,23	0,00	0,00	0,00	0,00	0,31
	500	–	500	500	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	503	–	503	503	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	504	–	504	504	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	999	–	999	999	0,49	0,29	0,09	0,00	0,07	0,03	0,23

Objektgrößen Nutzdaten

Mimetype		Verteilung	T	$\hat{F}_{Req}(T)$	$\hat{F}_{Vol}(T)$	D_{KS}	D_A
Application	Body	lognormal	282.497	70,95%	16,63%	0,053	0,002
	Tail	weibull					
Audio	Body	pareto	3.356.787	98,73%	26,39%	0,057	0,012
	Tail	pareto					
Image	Body	lognormal	99.851	98,59%	82,12%	0,003	0,008
	Tail	pareto					
Text	Body	lognormal	7.421	77,07%	38,56%	0,030	0,017
	Tail	pareto					
Video	Body	weibull	5.213.131	95,82%	56,23%	0,058	0,015
	Tail	pareto					
Other	Body	lognormal	233.027	93,53%	11,41%	0,052	0,009
	Tail	pareto					

Objektgrößen Kontrollnachrichten

Statuscode	Funktion	M	D_{KS}	D_A
302	normal	599	0,287	0,040
304	normal	343	0,157	0,046
404	pareto	12.827	0,239	0,020
500	normal	1.932	0,070	0,031
503	normal	897	0,342	0,088
504	normal	–	–	–
999	pareto	61.581	0,178	0,027

Datenraten HTTP-Responses mit Statuscodes 200 und 304

	Statuscode				Toplevel Domain	Verteilung	D_{KS}	D_A
	C_L-C_1	C_1-S	C_1-C_2	C_2-S				
Hit	200	–	–	–	com	weibull	0,036	0,014
					de	weibull	0,029	0,007
					net	weibull	0,016	0,008
					num	weibull	0,028	0,014
					edu	weibull	0,027	0,010
					org	weibull	0,019	0,009
					other	weibull	0,017	0,008
	304	–	–	–	com	weibull	0,053	0,013
					de	weibull	0,030	0,005
					other	weibull	0,040	0,006

Miss	200	200	–	–	com	weibull	0,020	0,010
					de	lognormal	0,017	0,006
					net	weibull	0,014	0,007
					num	weibull	0,009	0,004
					edu	weibull	0,017	0,006
					org	weibull	0,023	0,009
					other	weibull	0,024	0,010
	200	304	–	–	com	weibull	0,025	0,011
	304	200	–	–	com	weibull	0,030	0,010
					de	lognormal	0,026	0,009
					num	weibull	0,051	0,017
					other	weibull	0,022	0,009
Neighbor-Hit	200	–	200	–	com	weibull	0,023	0,009
					de	weibull	0,024	0,020
					net	weibull	0,025	0,031
					edu	weibull	0,030	0,010
					org	lognormal	0,026	0,009
					other	weibull	0,051	0,017
	200	–	304	–	com	weibull	0,017	0,014
	304	–	200	–	com	weibull	0,017	0,010
					de	weibull	0,023	0,006
					net	weibull	0,024	0,009
					other	lognormal	0,053	0,020
	304	–	304	–	com	weibull	0,030	0,031
Neighbor-Miss	200	–	200	200	com	lognormal	0,040	0,020
					de	weibull	0,051	0,031
					net	weibull	0,017	0,006
					edu	weibull	0,017	0,009
					org	lognormal	0,023	0,014
					other	lognormal	0,053	0,010
	304	–	200	200	com	weibull	0,030	0,014
					de	lognormal	0,040	0,009
					other	weibull	0,025	0,011

Datenraten Kontrollnachrichten

	Statuscode				Toplevel Domain	Verteilung	D_{KS}	D_A
	C_L-C_1	C_1-S	C_1-C_2	C_2-S				
Miss	302	302	–	–	com	weibull	0,015	0,005
					net	weibull	0,129	0,028
					num	lognormal	0,112	0,018
	404	404	–	–	com	weibull	0,018	0,008
	500	500	–	–	num	lognormal	0,020	0,006
Neighbor--Miss	302	–	302	302	com	weibull	0,017	0,003
					de	weibull	0,013	0,003
					net	lognormal	0,028	0,014
	404	–	404	404	com	weibull	0,027	0,010
					de	lognormal	0,019	0,009
					other	weibull	0,017	0,008
	999	–	999	999	com	weibull	0,017	0,003
					de	lognormal	0,013	0,003
					net	weibull	0,016	0,003
					edu	lognormal	0,018	0,007
					org	weibull	0,022	0,009
					other	weibull	0,036	0,013

C.2 Datensatz 98: 1. Februar 1998 – 28. Februar 1998

Zwischenankunftszeiten in mittlerer Hauptverkehrsstunde

DFN-Cache-Server	HTTP-Responses in HvStd über 20 Tage	mittlere Ankunftsrate [HTTP-Requests/s]
berlin	598.483	8,312
frankfurt	771.709	10,718
hamburg	820.891	11,401
hannover	582.396	8,089
karlsruhe	854.208	11,864
koeln	853.026	11,848
leipzig	854.662	11,870
muenchen	732.996	10,180
nuernberg	616.843	8,567
stuttgart	712.234	9,892
Summe	**7.397.448**	**102,742**

Verteilung Toplevel Domains

	com	de	net	num	edu	org	other
HTTP-Requests	46,29	30,58	6,25	5,24	0,91	1,38	9,35

Verteilung Mimetypes über Toplevel Domains

	com	de	net	num	edu	org	other
Application	0,82	0,87	0,67	1,47	1,83	1,93	1,10
Audio	0,18	0,09	0,23	0,11	0,32	0,23	0,23
Image	74,56	69,09	59,20	77,28	66,92	69,14	73,37
Text	23,11	28,71	37,14	18,82	30,14	27,00	24,42
Video	0,09	0,02	0,12	0,18	0,09	0,09	0,07
Other	1,25	1,23	2,64	2,14	0,69	1,61	0,81

Verteilung HTTP-Codes über Toplevel Domains

	com	de	net	num	edu	org	other
200	77,91	70,96	69,48	77,10	100,00	100,00	87,02
302	4,03	1,28	15,75	0,00	0,00	0,00	0,00
304	16,40	26,92	11,31	17,89	0,00	0,00	12,98
404	1,19	0,84	0,00	0,00	0,00	0,00	0,00
500	0,00	0,00	0,00	0,00	0,00	0,00	0,00
503	0,00	0,00	0,00	0,00	0,00	0,00	0,00
504	0,00	0,00	0,00	0,00	0,00	0,00	0,00
other	0,46	0,00	3,46	5,01	0,00	0,00	0,00

Hit / Miss

	Statuscode				HTTP-Responses (in %)						
	C_L-C_1	C_1-S	C_1-C_2	C_2-S	com	de	net	num	edu	org	other
Hit	200	–	–	–	11,76	7,15	1,16	1,39	0,00	0,35	1,55
	304	–	–	–	2,94	2,68	0,36	0,42	0,00	0,00	0,36
Miss	200	200	–	–	19,07	11,63	2,68	2,20	0,88	1,00	5,75
	200	304	–	–	0,44	0,32	0,00	0,00	0,00	0,00	0,00
	304	200	–	–	3,38	4,85	0,33	0,49	0,00	0,00	0,82
	304	304	–	–	0,33	0,00	0,00	0,00	0,00	0,00	0,00
Neighbor-Hit	200	–	200	–	3,82	2,02	0,38	0,34	0,00	0,00	0,62
	200	–	304	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	200	–	0,74	0,47	0,00	0,00	0,00	0,00	0,00
	304	–	304	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Neighbor-Miss	200	–	200	200	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	200	–	200	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	200	–	304	200	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	200	–	304	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	200	200	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	200	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	304	200	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	304	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Hit	302	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	404	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	500	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	503	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	504	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	999	–	–	–	0,00	0,00	0,21	0,00	0,00	0,00	0,00
Miss	302	302	–	–	1,82	0,38	0,96	0,00	0,00	0,00	0,00
	404	404	–	–	0,54	0,25	0,00	0,00	0,00	0,00	0,00
	500	500	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	503	503	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	504	504	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	999	999	–	–	0,21	0,00	0,00	0,26	0,00	0,00	0,00
Neighbor-Hit	302	–	302	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	404	–	404	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	500	–	500	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	503	–	503	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	504	–	504	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	999	–	999	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Neighbor-Miss	302	–	302	302	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	404	–	404	404	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	500	–	500	500	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	503	–	503	503	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	504	–	504	504	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	999	–	999	999	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Objektgrößen Nutzdaten

Mimetype		Verteilung	T	$\hat{F}_{Req}(T)$	$\hat{F}_{Vol}(T)$	D_{KS}	D_A
Application	Body	pareto	299.777	76,90%	18,71%	0,050	0,004
	Tail	weibull					
Audio	Body	pareto	3.487.634	96,92%	29,62%	0,009	0,001
	Tail	pareto					
Image	Body	lognormal	95.804	99,31%	75,12%	0,025	0,004
	Tail	pareto					
Text	Body	lognormal	11.913	85,05%	40,23%	0,021	0,017
	Tail	pareto					
Video	Body	weibull	5.307.315	96,68%	69,55%	0,051	0,015
	Tail	pareto					
Other	Body	lognormal	958.453	91,16%	8,50%	0,139	0,047
	Tail	pareto					

Objektgrößen Kontrollnachrichten

Statuscode	Funktion	M	D_{KS}	D_A
302	normal	709	0,214	0,042
304	normal	413	0,152	0,045
404	pareto	32.668	0,137	0,032
500	normal	1.989	0,064	0,032
503	normal	1.204	0,214	0,091
504	normal	–	–	–
999	pareto	25.859	0,179	0,047

Datenraten HTTP-Responses mit Statuscodes 200 und 304

	Statuscode				Toplevel Domain	Verteilung	D_{KS}	D_A
	C_L-C_1	C_1-S	C_1-C_2	C_2-S				
Hit	200	–	–	–	com	lognormal	0,025	0,006
					de	lognormal	0,008	0,002
					net	lognormal	0,020	0,006
					num	lognormal	0,017	0,003
					org	lognormal	0,013	0,003
					other	lognormal	0,016	0,003
	304	–	–	–	com	lognormal	0,018	0,007
					de	lognormal	0,022	0,009
					net	lognormal	0,036	0,013
					num	lognormal	0,010	0,004
					other	lognormal	0,016	0,005

Miss	200	200	–	–	com	lognormal	0,024	0,011
					de	lognormal	0,010	0,005
					net	weibull	0,023	0,010
					num	weibull	0,021	0,009
					edu	lognormal	0,020	0,010
					org	lognormal	0,017	0,008
					other	lognormal	0,009	0,003
	200	304	–	–	com	lognormal	0,010	0,002
					de	lognormal	0,006	0,002
	304	200	–	–	com	lognormal	0,016	0,007
					de	lognormal	0,010	0,003
					net	lognormal	0,019	0,009
					num	lognormal	0,015	0,004
					other	lognormal	0,012	0,006
	304	304	–	–	com	lognormal	0,062	0,007
Neighbor-Hit	200	–	200	–	com	weibull	0,021	0,004
					de	lognormal	0,020	0,002
					net	lognormal	0,019	0,007
					num	lognormal	0,009	0,003
					other	weibull	0,021	0,002
	304	–	200	–	com	weibull	0,020	0,002
					de	lognormal	0,019	0,009

Datenraten Kontrollnachrichten

	Statuscode				**Toplevel Domain**	**Verteilung**	D_{KS}	D_A
	C_L-C_1	**C_1-S**	**C_1-C_2**	**C_2-S**				
Hit	999	–	–	–	net	lognormal	0,058	0,014
Miss	302	302	–	–	com	lognormal	0,024	0,009
					de	lognormal	0,015	0,005
					net	lognormal	0,041	0,012
	404	404	–	–	com	lognormal	0,019	0,007
					de	lognormal	0,009	0,004
	999	999	–	–	com	lognormal	0,008	0,003
					num	weibull	0,048	0,011

C.3 Datensatz 99: 1. Februar 1999 – 28. Februar 1999

Zwischenankunftszeiten in mittlerer Hauptverkehrsstunde

DFN-Cache-Server	HTTP-Responses in HvStd über 20 Tage	mittlere Ankunftsrate [HTTP-Requests/s]
berlin	859.552	11,938
frankfurt	916.094	12,724
hamburg	1.034.923	14,374
hannover	681.616	9,467
karlsruhe	1.019.015	14,153
koeln	1.110.019	15,417
leipzig	939.545	13,049
muenchen	1.196.102	16,613
nuernberg	792.913	11,013
stuttgart	820.853	11,401
Summe	**9.370.632**	**130,148**

Verteilung Toplevel Domains

	com	de	net	num	edu	org	other
HTTP-Requests	47,60	31,91	7,32	3,85	0,53	1,16	7,63

Verteilung Mimetypes über Toplevel Domains

	com	de	net	num	edu	org	other
Application	1,27	1,01	0,78	4,25	1,83	1,30	1,29
Audio	0,17	0,09	0,13	0,21	0,41	0,20	0,23
Image	75,49	72,31	71,08	75,92	66,39	70,86	75,06
Text	22,28	26,02	27,61	17,92	30,81	26,91	22,77
Video	0,10	0,01	0,10	0,16	0,13	0,04	0,05
Other	0,69	0,55	0,30	1,54	0,43	0,69	0,60

Verteilung HTTP-Codes über Toplevel Domains

	com	de	net	num	edu	org	other
200	74,95	73,34	76,36	80,00	100,00	100,00	85,95
302	5,38	1,13	10,87	0,00	0,00	0,00	0,00
304	16,67	24,39	12,37	17,45	0,00	0,00	13,64
404	1,17	1,01	0,00	0,00	0,00	0,00	0,00
500	0,19	0,00	0,00	0,47	0,00	0,00	0,00
503	0,25	0,00	0,00	0,00	0,00	0,00	0,00
504	0,26	0,13	0,40	2,07	0,00	0,00	0,41
other	1,13	0,00	0,00	0,00	0,00	0,00	0,00

Hit / Miss

	Statuscode				HTTP-Responses (in %)						
	C_L-C_1	C_1-S	C_1-C_2	C_2-S	com	de	net	num	edu	org	other
Hit	200	–	–	–	10,87	7,45	1,74	0,84	0,00	0,26	1,11
	304	–	–	–	3,53	3,47	0,49	0,39	0,00	0,00	0,33
Miss	200	200	–	–	19,44	12,91	3,13	1,94	0,51	0,86	4,70
	200	304	–	–	0,75	0,47	0,00	0,00	0,00	0,00	0,00
	304	200	–	–	2,74	2,95	0,39	0,27	0,00	0,00	0,68
	304	304	–	–	0,61	0,58	0,00	0,00	0,00	0,00	0,00
Neighbor-Hit	200	–	200	–	3,55	1,86	0,56	0,21	0,00	0,00	0,49
	200	–	304	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	200	–	0,82	0,54	0,00	0,00	0,00	0,00	0,00
	304	–	304	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Neighbor-Miss	200	–	200	200	0,00	0,00	0,00	0,00	0,00	0,00	0,07
	200	–	200	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	200	–	304	200	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	200	–	304	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	200	200	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	200	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	304	200	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	304	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Hit	302	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	404	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	500	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	503	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	504	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	999	–	–	–	0,26	0,00	0,00	0,00	0,00	0,00	0,00
Miss	302	302	–	–	2,49	0,35	0,77	0,00	0,00	0,00	0,00
	404	404	–	–	0,54	0,31	0,00	0,00	0,00	0,00	0,00
	500	500	–	–	0,09	0,00	0,00	0,02	0,00	0,00	0,00
	503	503	–	–	0,12	0,00	0,00	0,00	0,00	0,00	0,00
	504	504	–	–	0,12	0,04	0,03	0,08	0,00	0,00	0,03
	999	999	–	–	0,26	0,00	0,00	0,00	0,00	0,00	0,00
Neighbor-Hit	302	–	302	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	404	–	404	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	500	–	500	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	503	–	503	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	504	–	504	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	999	–	999	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Neighbor-Miss	302	–	302	302	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	404	–	404	404	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	500	–	500	500	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	503	–	503	503	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	504	–	504	504	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	999	–	999	999	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Objektgrößen Nutzdaten

Mimetype		Verteilung	T	$\hat{F}_{Req}(T)$	$\hat{F}_{Vol}(T)$	D_{KS}	D_A
Application	Body	pareto	535.666	88,49%	14,99%	0,045	0,005
	Tail	weibull					
Audio	Body	pareto	3.133.981	94,82%	26,83%	0,062	0,007
	Tail	pareto					
Image	Body	lognormal	108.902	99,61%	71,08%	0,047	0,002
	Tail	pareto					
Text	Body	lognormal	76.113	99,06%	50,87%	0,026	0,011
	Tail	pareto					
Video	Body	weibull	2.322.010	89,08%	43,76%	0,039	0,015
	Tail	pareto					
Other	Body	lognormal	37.166	89,95%	10,05%	0,072	0,028
	Tail	weibull					

Objektgrößen Kontrollnachrichten

Statuscode	Funktion	M	D_{KS}	D_A
302	normal	863	0,125	0,027
304	normal	464	0,136	0,036
404	pareto	75.871	0,149	0,026
500	normal	2.992	0,286	0,073
503	normal	2.365	0,107	0,026
504	normal	2.548	0,128	0,036
999	pareto	75.803	0,138	0,040

Datenraten HTTP-Responses mit Statuscodes 200 und 304

	Statuscode				Toplevel Domain	Verteilung	D_{KS}	D_A
	C_L-C_1	C_1-S	C_1-C_2	C_2-S				
Hit	200	–	–	–	com	lognormal	0,024	0,010
					de	lognormal	0,059	0,032
					net	lognormal	0,068	0,036
					num	lognormal	0,050	0,025
					org	lognormal	0,056	0,034
					other	lognormal	0,057	0,034
	304	–	–	–	com	lognormal	0,034	0,004
					de	lognormal	0,114	0,049
					net	lognormal	0,106	0,047
					num	lognormal	0,106	0,046
					other	lognormal	0,106	0,050

Miss	200	200	–	–	com	lognormal	0,019	0,008
					de	lognormal	0,014	0,005
					net	lognormal	0,014	0,005
					num	lognormal	0,009	0,004
					edu	lognormal	0,005	0,002
					org	lognormal	0,007	0,002
					other	lognormal	0,006	0,002
	200	304	–	–	com	lognormal	0,024	0,011
					de	lognormal	0,034	0,016
	304	200	–	–	com	lognormal	0,024	0,007
					de	lognormal	0,067	0,036
					net	lognormal	0,045	0,023
					num	lognormal	0,056	0,032
					other	lognormal	0,050	0,025
	304	304	–	–	com	weibull	0,035	0,015
					de	lognormal	0,068	0,034
Neighbor-Hit	200	–	200	–	com	lognormal	0,014	0,005
					de	lognormal	0,009	0,004
					net	weibull	0,005	0,002
					num	lognormal	0,007	0,025
					other	weibull	0,047	0,015
	200	–	200	–	com	lognormal	0,024	0,011
					de	lognormal	0,024	0,007
Neighbor-Miss	200	–	200	200	other	weibull	0,067	0,036

Datenraten Kontrollnachrichten

	Statuscode				**Toplevel Domain**	**Verteilung**	D_{KS}	D_A
	C_L-C_1	**C_1-S**	**C_1-C_2**	**C_2-S**				
Hit	999	–	–	–	com	weibull	0,090	0,022
Miss	302	302	–	–	com	lognormal	0,043	0,012
					de	lognormal	0,045	0,025
					net	lognormal	0,048	0,023
	404	404	–	–	com	lognormal	0,013	0,007
					de	lognormal	0,047	0,021
	500	500	–	–	com	weibull	0,076	0,023
					num	lognormal	0,071	0,023
	503	503	–	–	com	weibull	0,059	0,014
	504	504	–	–	com	Timeout 300 Sekunden		
					de			
					net			
					num			
					other			
	999	999	–	–	com	lognormal	0,014	0,007

C.4 Datensatz 00: 1. Februar 2000 – 28. Februar 2000

Zwischenankunftszeiten in mittlerer Hauptverkehrsstunde

DFN-Cache-Server	HTTP-Responses in HvStd über 20 Tage	mittlere Ankunftsrate [HTTP-Requests/s]
berlin	679.332	9,435
frankfurt	778.665	10,815
hamburg	1.113.708	15,468
hannover	614.589	8,536
karlsruhe	1.067.013	14,820
koeln	1.276.232	17,725
leipzig	1.337.670	18,579
muenchen	1.501.160	20,849
nuernberg	681.796	9,469
stuttgart	629.987	8,750
Summe	**9.680.152**	**134,447**

Verteilung Toplevel Domains

	com	de	net	num	edu	org	other
HTTP-Requests	40,63	40,39	8,06	2,88	0,34	0,85	6,85

Verteilung Mimetypes über Toplevel Domains

	com	de	net	num	edu	org	other
Application	1,83	1,90	1,72	2,63	4,31	1,85	2,36
Audio	0,16	0,07	0,10	0,42	0,16	0,21	0,30
Image	73,56	73,32	71,55	78,01	68,44	72,15	72,13
Text	23,17	23,40	25,68	17,44	26,34	24,41	23,62
Video	0,18	0,02	0,13	0,10	0,09	0,24	0,10
Other	1,10	1,30	0,81	1,41	0,67	1,15	1,48

Verteilung HTTP-Codes über Toplevel Domains

	com	de	net	num	edu	org	other
200	74,05	69,42	70,33	79,87	100,00	98,58	77,58
302	7,07	1,45	11,45	0,00	0,00	0,00	4,55
304	14,15	22,70	13,88	18,91	0,00	0,00	11,25
404	1,12	1,16	0,00	0,00	0,00	0,00	0,00
500	0,18	0,10	0,00	0,00	0,00	0,00	0,00
503	0,18	0,11	0,17	1,21	0,00	1,42	0,27
504	1,22	4,36	4,17	0,00	0,00	0,00	6,36
other	2,03	0,70	0,00	0,00	0,00	0,00	0,00

Hit / Miss

	Statuscode				HTTP-Responses (in %)						
	C_L-C_1	C_1-S	C_1-C_2	C_2-S	com	de	net	num	edu	org	other
Hit	200	–	–	–	5,09	6,51	1,41	0,53	0,00	0,00	0,56
	304	–	–	–	1,44	2,85	0,58	0,27	0,00	0,00	0,00
Miss	200	200	–	–	21,19	18,38	3,59	1,53	0,33	0,81	4,32
	200	304	–	–	0,25	0,29	0,00	0,00	0,00	0,00	0,00
	304	200	–	–	2,86	4,53	0,50	0,26	0,00	0,00	0,75
	304	304	–	–	0,33	0,43	0,00	0,00	0,00	0,00	0,00
Neighbor-Hit	200	–	200	–	2,59	2,00	0,49	0,16	0,00	0,00	0,27
	200	–	304	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	200	–	0,94	1,08	0,00	0,00	0,00	0,00	0,00
	304	–	304	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Neighbor-Miss	200	–	200	200	0,06	0,00	0,00	0,00	0,00	0,00	0,00
	200	–	200	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	200	–	304	200	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	200	–	304	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	200	200	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	200	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	304	200	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	304	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Hit	302	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	404	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	500	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	503	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	504	–	–	–	0,46	1,70	0,33	0,00	0,00	0,00	0,42
	999	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Miss	302	302	–	–	2,79	0,57	0,89	0,00	0,00	0,00	0,30
	404	404	–	–	0,44	0,45	0,00	0,00	0,00	0,00	0,00
	500	500	–	–	0,04	0,04	0,00	0,00	0,00	0,00	0,00
	503	503	–	–	0,10	0,04	0,01	0,03	0,00	0,01	0,02
	504	504	–	–	0,02	0,01	0,00	0,00	0,00	0,00	0,00
	999	999	–	–	0,80	0,27	0,00	0,00	0,00	0,00	0,00
Neighbor-Hit	302	–	302	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	404	–	404	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	500	–	500	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	503	–	503	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	504	–	504	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	999	–	999	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Neighbor-Miss	302	–	302	302	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	404	–	404	404	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	500	–	500	500	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	503	–	503	503	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	504	–	504	504	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	999	–	999	999	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Objektgrößen Nutzdaten

Mimetype		Verteilung	T	$\hat{F}_{Req}(T)$	$\hat{F}_{Vol}(T)$	D_{KS}	D_A
Application	Body	pareto	2.918.378	96,03%	15,23%	0,047	0,008
	Tail	pareto					
Audio	Body	pareto	3.260.490	94,03%	32,12%	0,086	0,012
	Tail	pareto					
Image	Body	lognormal	135.677	99,79%	76,12%	0,074	0,008
	Tail	pareto					
Text	Body	weibull	57.786	98,52%	44,38%	0,033	0,003
	Tail	pareto					
Video	Body	weibull	5.405.306	97,64%	52,82%	0,144	0,035
	Tail	pareto					
Other	Body	lognormal	51.957	94,60%	12,92%	0,078	0,019
	Tail	weibull					

Objektgrößen Kontrollnachrichten

Statuscode	Funktion	M	D_{KS}	D_A
302	normal	1.119	0,086	0,022
304	normal	520	0,136	0,060
404	pareto	81.249	0,087	0,037
500	normal	2.927	0,261	0,156
503	normal	2.624	0,069	0,020
504	normal	2.875	0,181	0,036
999	pareto	38.974	0,180	0,041

Datenraten HTTP-Responses mit Statuscodes 200 und 304

	Statuscode				Toplevel Domain	Verteilung	D_{KS}	D_A
	C_L-C_1	C_1-S	C_1-C_2	C_2-S				
Hit	200	–	–	–	com	weibull	0,021	0,007
					de	lognormal	0,065	0,030
					net	lognormal	0,061	0,031
					num	lognormal	0,066	0,031
					other	lognormal	0,064	0,030
	304	–	–	–	com	weibull	0,028	0,008
					de	lognormal	0,119	0,045
					net	lognormal	0,113	0,046
					num	lognormal	0,123	0,049

Miss	200	200	–	–	com	weibull	0,020	0,008
					de	lognormal	0,026	0,006
					net	lognormal	0,028	0,006
					num	lognormal	0,019	0,005
					edu	lognormal	0,022	0,012
					org	lognormal	0,023	0,008
					other	lognormal	0,021	0,005
	200	304	–	–	com	weibull	0,028	0,010
					de	lognormal	0,035	0,016
	304	200	–	–	com	weibull	0,025	0,010
					de	lognormal	0,080	0,038
					net	lognormal	0,086	0,039
					num	lognormal	0,086	0,040
					other	lognormal	0,082	0,037
	304	304	–	–	com	weibull	0,025	0,010
					de	lognormal	0,093	0,043
Neighbor-Hit	200	–	200	–	com	lognormal	0,092	0,049
					de	weibull	0,027	0,011
					net	lognormal	0,021	0,005
					num	weibull	0,028	0,012
					other	lognormal	0,035	0,008
	304	–	200	–	com	lognormal	0,025	0,010
					de	weibull	0,156	0,066
Neighbor-Miss	200	–	200	200	com	weibull	0,219	0,108

Datenraten Kontrollnachrichten

	Statuscode				Toplevel Domain	Verteilung	D_{KS}	D_A
	C_L-C_1	C_1-S	C_1-C_2	C_2-S				
Hit	504	504	–	–	com	lognormal	0,052	0,017
					de	lognormal	0,086	0,042
					net	lognormal	0,084	0,042
					other	lognormal	0,086	0,037
Miss	302	302	–	–	com	weibull	0,021	0,013
					de	lognormal	0,084	0,038
					net	lognormal	0,098	0,049
					other	lognormal	0,092	0,049
	404	404	–	–	com	weibull	0,027	0,011
					de	lognormal	0,066	0,025
	500	500	–	–	com	weibull	0,066	0,009
					de	lognormal	0,263	0,082
	503	503	–	–	com	lognormal	0,343	0,170
					de	weibull	0,156	0,066
					net	weibull	0,219	0,108
					num	lognormal	0,328	0,152
					org	weibull	0,188	0,070
					other	weibull	0,172	0,071
	504	504	–	–	com	Timeout 300 Sekunden		
					de			
					other			
	999	999	–	–	com	weibull	0,027	0,015
					de	lognormal	0,087	0,037

C.5 Datensatz 01: 14. November 2000 – 12. Dezember 2000

Zwischenankunftszeiten in mittlerer Hauptverkehrsstunde

DFN-Cache-Server	HTTP-Responses in HvStd über 20 Tage	mittlere Ankunftsrate [HTTP-Requests/s]
cs-han10	3.927.475	54,548
cs-han11	6.329.640	87,912
cs-han20	7.668.536	106,507
cs-han21	3.404.754	47,288
Summe	**35.161.170**	**296,256**

Verteilung Toplevel Domains

	com	de	net	num	edu	org	other
HTTP-Requests	37,05	43,68	7,56	2,91	0,28	0,85	7,66

Verteilung Mimetypes über Toplevel Domains

	com	de	net	num	edu	org	other
Application	2,88	4,14	3,08	5,31	1,86	2,08	3,18
Audio	0,16	0,08	0,15	0,37	0,16	0,29	0,14
Image	73,38	74,97	77,84	72,71	70,74	74,89	76,85
Text	20,81	20,12	14,35	16,82	26,58	21,28	18,74
Video	0,18	0,03	0,23	3,28	0,09	0,11	0,17
Other	2,60	0,67	4,36	1,52	0,57	1,34	0,91

Verteilung HTTP-Codes über Toplevel Domains

	com	de	net	num	edu	org	other
200	70,86	67,80	71,69	72,98	100,00	99,72	75,53
302	6,54	1,95	5,27	0,00	0,00	0,00	0,00
304	19,23	28,11	22,98	26,13	0,00	0,00	22,77
404	1,02	1,02	0,00	0,00	0,00	0,00	1,44
500	0,07	0,16	0,00	0,15	0,00	0,00	0,12
503	0,09	0,02	0,00	0,35	0,00	0,00	0,00
504	0,08	0,04	0,06	0,38	0,00	0,28	0,14
other	2,12	0,91	0,00	0,00	0,00	0,00	0,00

Hit / Miss

	Statuscode				HTTP-Responses (in %)						
	C_L-C_1	C_1-S	C_1-C_2	C_2-S	com	de	net	num	edu	org	other
Hit	200	–	–	–	7,86	8,82	1,60	0,61	0,00	0,20	1,30
	304	–	–	–	3,80	5,85	0,88	0,39	0,00	0,00	0,77
Miss	200	200	–	–	16,40	18,21	3,42	1,45	0,27	0,63	4,04
	200	304	–	–	1,22	1,69	0,29	0,00	0,00	0,00	0,33
	304	200	–	–	2,47	4,88	0,83	0,35	0,00	0,00	0,68
	304	304	–	–	0,70	1,31	0,00	0,00	0,00	0,00	0,27
Neighbor-Hit	200	–	200	–	0,25	0,30	0,00	0,00	0,00	0,00	0,00
	200	–	304	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	200	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	304	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Neighbor-Miss	200	–	200	200	0,00	0,00	0,00	0,02	0,00	0,00	0,00
	200	–	200	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	200	–	304	200	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	200	–	304	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	200	200	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	200	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	304	200	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	304	–	304	304	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Hit	302	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	404	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	500	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	503	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	504	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	999	–	–	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Miss	302	302	–	–	2,37	0,84	0,39	0,00	0,00	0,00	0,00
	404	404	–	–	0,37	0,44	0,00	0,00	0,00	0,00	0,11
	500	500	–	–	0,03	0,07	0,00	0,00	0,00	0,00	0,01
	503	503	–	–	0,03	0,01	0,00	0,01	0,00	0,00	0,00
	504	504	–	–	0,03	0,02	0,00	0,01	0,00	0,00	0,01
	999	999	–	–	0,77	0,39	0,00	0,00	0,00	0,00	0,00
Neighbor-Hit	302	–	302	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	404	–	404	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	500	–	500	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	503	–	503	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	504	–	504	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	999	–	999	–	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Neighbor-Miss	302	–	302	302	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	404	–	404	404	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	500	–	500	500	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	503	–	503	503	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	504	–	504	504	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	999	–	999	999	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Objektgrößen Nutzdaten

Mimetype		Verteilung	T	$\hat{F}_{Req}(T)$	$\hat{F}_{Vol}(T)$	D_{KS}	D_A
Application	Body	Pareto	2.907.935	97,59%	20,83%	0,020	0,003
	Tail	Pareto					
Audio	Body	Pareto	3.216.056	92,16%	25,40%	0,169	0,014
	Tail	Pareto					
Image	Body	Lognormal	69.100	99,15%	80,23%	0,025	0,001
	Tail	Pareto					
Text	Body	Weibull	82.142	98,95%	56,02%	0,015	0,001
	Tail	Pareto					
Video	Body	Weibull	10.038.097	97,49%	62,17%	0,100	0,032
	Tail	Pareto					
Other	Body	Lognormal	126.457	91,53%	4,85%	0,061	0,022
	Tail	Weibull					

Objektgrößen Kontrollnachrichten

Statuscode	Funktion	M	D_{KS}	D_A
302	normal	1.161	0,049	0,013
304	normal	516	0,221	0,044
404	pareto	88.639	0,110	0,034
500	normal	3.007	0,401	0,092
503	normal	2.213	0,097	0,034
504	normal	2.258	0,172	0,046
999	pareto	46.469	0,106	0,037

Datenraten HTTP-Responses mit Statuscodes 200 und 304

	Statuscode				Toplevel Domain	Verteilung	D_{KS}	D_A
	C_L-C_1	C_1-S	C_1-C_2	C_2-S				
Hit	200	–	–	–	com	lognormal	0,027	0,011
					de	lognormal	0,023	0,011
					net	weibull	0,033	0,008
					num	lognormal	0,019	0,010
					org	lognormal	0,010	0,005
					other	lognormal	0,014	0,007
	304	–	–	–	com	lognormal	0,052	0,023
					de	lognormal	0,053	0,023
					net	lognormal	0,046	0,022
					num	lognormal	0,059	0,023
					other	lognormal	0,055	0,020

Miss	200	200	–	–	com	weibull	0,024	0,014
					de	lognormal	0,017	0,008
					net	lognormal	0,027	0,013
					num	weibull	0,025	0,014
					edu	weibull	0,021	0,013
					org	lognormal	0,018	0,008
					other	lognormal	0,026	0,009
	200	304	–	–	com	lognormal	0,011	0,004
					de	lognormal	0,018	0,006
					net	weibull	0,022	0,012
					other	lognormal	0,019	0,007
	304	200	–	–	com	weibull	0,055	0,013
					de	weibull	0,020	0,011
					net	weibull	0,048	0,017
					num	weibull	0,029	0,014
					other	weibull	0,013	0,007
	304	304	–	–	com	lognormal	0,055	0,017
					de	weibull	0,037	0,019
					other	lognormal	0,042	0,015
Neighbor-Hit	200	–	200	–	com	lognormal	0,012	0,006
					de	lognormal	0,027	0,013
Neighbor-Miss	200	–	200	200	num	lognormal	0,014	0,006

Datenraten Kontrollnachrichten

	Statuscode				**Toplevel Domain**	**Verteilung**	D_{KS}	D_A
	C_L-C_1	**C_1-S**	**C_1-C_2**	**C_2-S**				
Miss	302	302	–	–	com	lognormal	0,079	0,019
					de	weibull	0,027	0,015
					net	lognormal	0,102	0,033
	404	404	–	–	com	lognormal	0,069	0,015
					de	weibull	0,026	0,009
					other	weibull	0,037	0,011
	500	500	–	–	com	weibull	0,046	0,023
					de	weibull	0,027	0,012
					num	weibull	0,128	0,043
					other	weibull	0,051	0,022
	503	503	–	–	com	lognormal	0,049	0,013
					de	weibull	0,052	0,011
					num	lognormal	0,097	0,041
	504	504	–	–	com	Timeout 300 Sekunden		
					de			
					net			
					num			
					org			
					other			
	999	999	–	–	com	weibull	0,079	0,032
					de	lognormal	0,079	0,019

Literaturverzeichnis

[AAY97] J. M. Almeida, V. Almeida, D. J. Yates. *Measuring the Behavior of a World-Wide Web Server.* Proc. 7th IFIP Conference on Performance Networking, S. 57–72, White Plains, April 1997.

[Abd98] G. Abdulla. *Analysis and Modeling of World Wide Web Traffic.* Ph.D. thesis, Computer Science Department, Virginia Tech., 1998.

[ACGM+01] A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, S. Raghavan. *Searching the Web.* ACM Transactions on Internet Technology, Vol. 1, Nr. 1, S. 2–43, August 2001.

[AFJ98] M. Arlitt, R. Friedrich, T. Jin. *Performance Evaluation of Web Proxy Cache Replacement Policies.* Proc. 10th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation, Palma de Mallorca, S. 193–206, September 1998.

[And+93] M. Andreessen et. al. *XMosaic.* Februar 1993 http://archive.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html, Dezember 2001.

[APS99] M. Allman, V. Paxson, W. Stevens. *TCP Congestion Control.* RFC 2581, IETF, April 1999.

[ArWi97] M. F. Arlitt, C. L. Williamson. *Internet Web servers: Workload characterization and implications.* IEEE/ACM Transactions on Networking, Vol. 5, Nr. 5, S. 631–644, Oktober 1997.

[Aya+01] J. Ayars et. al. *Synchronized Multimedia Integration Language (SMIL 2.0).* World Wide Web Consortium, August 2001.

[Bac86] M. J. Bach. *Design of the Unix Operating System.* Prentice Hall, 1986.

[BaCr98] P. Barford, M. E. Crovella. *Generating Representative Web Workloads for Network and Server Performance Evaluation.* Proc. ACM SIGMETRICS '98, Madison, Juni 1998.

[BaCr99] P. Barford, M. E. Crovella. *A Performance Evaluation of Hyper Text Transfer Protocols.* Proc. ACM SIGMETRICS '99, Atlanta, Mai 1999.

[BAJ00] A.-L. Barabasi, R. Albert, H. Jeong. *Scale-free characteristics of random networks: The topology of the World Wide Web.* Physica A 281, S. 69–77, Mai 2000.

[BaOb00] G. Barish, K. Obraczka. *World Wide Web Caching: Trends and Techniques.* IEEE Communications Magazine, Vol. 38, Nr. 5, S. 178–184, Mai 2000.

[BBBC99] P. Barford, A. Bestavros, A. Bradley, M. E. Crovella. *Changes in Web Client Access Patterns: Characteristics and Caching Implications.* World Wide Web: Special Issue on Characterization and Performance Evaluation, Oktober 1999.

[BCF+98] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker. *On the Implications of Zipf's Law for Web.* Proc. 3rd International WWW Caching Workshop, Juni 1998.

[BDH+94] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, M. F. Schwartz. *The Harvest Information Discovery and Access System.* Proc. 2nd International WWW Conference, Chicago, Oktober 1994.

[BDR97] G. Banga, P. Druschel. *Measuring the Capacity of a Web Server.* Proc. 1st USENIX Symposium on Internet Technologies and Systems, Monterey, Dezember 1997.

[BDM98] G. Banga, P. Druschel, J. C. Mogul. *Better operating system features for faster network servers*. Proc. 1st Workshop on Internet Server Performance, Madison, Juni 1998.

[BDM99] G. Banga, J. C. Mogul, P. Druschel. *A Scalable and Explicit Event Delivery Mechanism for UNIX.* Proc. USENIX Annual Technical Conference, Monterey, Juni 1999.

[BGHP00] E. Bommaiaha, K. Guo, M. Hofmann, S. Paul. *Design and Implementation of a Caching System for Streaming Media over the Internet.* Proc. IEEE Real-Time Technology and Applications Symposium, Washington D.C., Juni 2000.

[BHW89] K. Burg, H. Haf, F. Wille. *Höhere Mathematik für Ingenieure, Band I Analysis.* Teubner, 2. Aufl., Stuttgart 1989.

[BKM+00] A. Broder, R. Kumar, F. Maghoul1, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, J. Wiener. *Graph structure in the web.* Proc. 9th International World Wide Web Conference, Amsterdam, Mai 2000.

[BL92] T. Berners-Lee. *The HTTP Protocol As Implemented In W3.* www-talk mailing list, 9. Januar 1992.

[BL94] T. Berners-Lee. *Uniform Resource Identifiers in WWW.* RFC 1630, IETF, Juni 1994.

[BLFF96] T. Berners-Lee, R. Fielding, H. Frystyk. *Hypertext Transfer Protocol – HTTP/1.0.* RFC 1945, IETF, Mai 1996.

[BLFi99] T. Berners-Lee, M. Fischetti. *Weaving the Web.* Harper, San Francisco, September 1999.

[BLFM98] T. Berners-Lee, R. Fielding, L. Masinter. *Uniform Resource Identifiers (URI): Generic Syntax.* RFC 2396, IETF, August 1998.

[BLMM94] T. Berners-Lee, L. Masinter, M. McCahill. *Uniform Resource Locators (URL).* RFC 1738, IETF, Dezember 1994.

[Blo70] B. Bloom. *Space/time Trade-offs in hash coding with allowable errors.* Communications of the ACM, Vol. 13, Nr. 7, S. 422–426, Juli 1970.

[BMS96] M. Baentsch, G. Molter, P. Sturm. *Introducing Application-level Replication and Naming into today's Web.* Proc. 5th International World Wide Web Conference, Paris, Mai 1996.

[BoFr93] N. Borenstein, N. Freed. *MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies.* RFC 1521, IETF, September 1993.

[BP+98] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler. *Extensible Markup Language (XML) 1.0.* World Wide Web Consortium, Oktober 2000.

[Bus00] M. Busari. *Simulation Evaluation of Web Caching Hierarchies.* M. Sc. Thesis, Department of Computer Science, University of Saskatchewan, Juni 2000.

[Cai96] R. Cailliau. *Warum erst heute? Das digitale Prinzip als treibende Kraft für eine vernetzte Informationsgesellschaft.* Macht Information – Konferenz über die Werte der Informationsgesellschaft. Bonn, September 1996.

[CCLS01] J. Cao, W. S. Cleveland, D. Lin, D. X. Sun. *On the Nonstationarity of Internet Traffic*. Proc. ACM SIGMETRICS 2001, Cambridge, Juni 2001.

[CCLS02] J. Cao, W. S. Cleveland, D. Lin, D. X. Sun. *Internet Traffic Tends Toward Poisson and Independent as the Load Increases.* To appear in C. Holmes, D. Denison, M. Hansen, B. Yu, B. Mallick. *Nonlinear Estimation and Classification.* Springer, 2002.

[CDF+98] R. Caceres, F. Douglis, A. Feldmann, G. Glass, M. Rabinovich. *Web Proxy Caching: The Devil is in the Details.* Proc. 1st Workshop on Internet Server Performance, Madison, Juni 1998.

[CDK+99] S. Chakrabarti, B. E. Dom, S. R. Kumar, P. Raghavan, S. Rajagopalan, A. Tomkins, D. Gibson, J. Kleinberg. *Mining the Web's link structure.* IEEE Computer, Vol. 32, Nr. 8, S. 60–67, August 1999.

[CDN+96] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, K. J. Worrell. *A Hierarchical Internet Object Cache.* USENIX 1996 Annual Technical Conference, San Diego, Januar 1996.

[CEB+00] A. Cerpa, J. Elson, H. Beheshti, A. Chankhunthod, P. Danzig, R. Jalan, C. Neerdaels, T. Schroeder, G. Tomlinson. *Network Elements Control Protocol.* Internet Draft, IETF. Expired August 2000.

[CFTW01] M. Cieslak, D. Forster, G. Tiwana, R. Wilson. *Web Cache Communication Protocol V2.0.* Internet Draft, IETF, Expires October 2001.

[Cha00] J. Charzinski. *HTTP/TCP Connection and Flow Characteristics.* Performance Evaluation Journal: Special Issue on Internet Performance Modelling, Vol. 42, Nr. 2/3, S. 149–162, September 2000.

[Che99] L. Chen. *Modellierung von Verkehrsflüssen in kooperativen Proxy-Verbünden.* Diplomarbeit, August 1999.

[CLDH02] I. Cooper, D. Li, M. Dahlin. M. Hamilton. *Requirements and Guidelines for A Resource Update Protocol.* Internet Draft, IETF. Expires August 2002.

[CLR00] M. Crovella, C. Lindemann, M. Reiser. *Internet Performance Modeling: The State of the Art at the Turn of the Century.* Performance Evaluation, 42, S. 91–108, 2000.

[CMT98] K. Claffy, G. Miller, K. Thomson. *The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone.* Proc. of Inet ’98 Conference. Juli 1998.

[CMT01] I. Cooper, I. Melve, G. Tomlinson. *Internet Web Replication and Caching Taxonomy.* RFC 3040, IETF, Januar 2001.

[Coc97] A. Cockroft. *How does Solaris 2.6 improve performance stats and Web performance?* Unix Insider, August 1997.

[CoKa01] E. Cohen, H. Kaplan. *Aging Through Cascaded Caches: Performance Issues in the Distribution of Web Content.* Proc. ACM SIGCOMM 2001, August 2001.

[CrBa98] M. Crovella, P. Barford. *The Network Effects of Prefetching.* Proc. IEEE Infocom '98, San Francisco, April 1998.

[CrBe97] M. E. Crovella, A. Bestavros. *Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes.* IEEE/ACM Transactions on Networking, Vol. 5, Nr. 6, S. 835–846, Dezember 1997.

[Cro82] D. H. Crocker, *Standard for The Format of ARPA Internet Text Messages.* STD 11, RFC 822, IETF, August 1982.

[CuSi99] D. E. Culler, J. P. Singh. *Parallel Computer Architecture - A Hardware/Software Approach.* Morgan Kaufman Publishers, San Francisco, 1999.

[DaMe97] R. Daniel, M. Mealling. *Resolution of Uniform Resource Identifiers using the Domain Name System.* RFC 2168, IETF, Juni 1997.

[Dan98] P. Danzig. *NetCache Architectur and Deployment.* Computer Networks and ISDN Systems, Vol. 30, S. 2081–2091, November 1998.

[DAPJ99] J. Dilley, M. Arlitt, S. Perret, T. Jin. *The Distributed Object Consistency Protocol.* HP Labs Technical Reports, HPL–1999–109, September 1999.

[Dav99] B. D. Davison. *Web Traffic Logs: An Imperfect Resource for Evaluation.* Proc. 9th Annual Conference of the Internet society, San Jose, Juni 1999.

[DeGl97] P. Denning, R. Glass (Hrsg.). *Before Memory was Virtual.* In the Beginning: Recollections of Software Pioneers, IEEE Press, 1997.

[DHS93] P. B. Danzig, R. Hall, M. F. Schwartz. *A Case for Caching File Objects Inside Internetworks.* Proc. ACM SIGCOMM '93, S. 239–248, San Francisco, September 1993.

[DiAl99] T. Dierks, C. Allen. *The TLS Protocol Version 1.0.* RFC 2246, IETF, Januar 1999.

[DiAr99] J. Dilley, M. Arlitt. *Improving Proxy Cache Performance: Analysis of Three Replacement Policies.* IEEE Internet Computing, Vol. 3, Nr. 6, S. 44–50, November 1999.

[DiPa95] A. Dingle, T. Partl. *Web Cache Coherence.* Proc. Fifth International World Wide Web Conference, Paris, Mai 1996.

[DJD99] S. G. Dykes, C. L. Jeffery, S. Das. *Taxonomy and design analysis for distributed Web caching.* Proc. 32nd Hawaii International Conference on System Sciences, Maui, Januar 1999.

[DMF97] B. M. Duska, D. Marwood, M. J. Feeley. *The Measured Access Characteristics of World-Wide-Web Client Proxy Caches.* Proc. 1st USENIX Symposium on Internet Technologies and Systems, Monterey, Dezember 1997.

[Dow01] A. B. Downey. *Evidence for long-tailed distributions in the Internet.* Proc. ACM SIGCOMM Internet Measurement Workshop 2001, San Francisco, November 2001.

[ESCK02] P. L'Ecuyer, R. Simard, E. J. Chen, W. D. Kelton. *An Objected-Oriented Random-Number Package with Many Long Streams and Substreams.* To appear INFORMS Operations Research, 2002.

[FCAB00] L. Fan, P. Cao, J. Almeida, A. Z. Broder. *Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol.* IEEE/ACM Transactions on Networking, Vol. 8, Nr. 3, S. 281–293, Juni 2000.

[FCD+99] A. Feldmann, R. Caceres, F. Douglis, G. Glass, M. Rabinovich. *Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments.* Proc. IEEE Infocom ’99, S. 107–116, 1999.

[FeWh98] A. Feldmann, W. Whitt. *Fitting Mixtures of Exponentials to Long-tail Distributions to Analyze Network Performance Models.* Performance evaluations, Vol. 31, 1998.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1.* RFC 2616, IETF, Juni 1999.

[FGSM01] M. J. Fischer, D. Gross, J. F. Shortle, D. M. Masi. *Analyzing the Waiting Time Process in Internet Queueing Systems with the Transform Approximation Method.* The Telecommunications Review, Vol. 11, S. 21–32, 2001.

[Fie95] R. Fielding, *Relative Uniform Resource Locators.* RFC 1808, IETF, Juni 1995.

[FJCL99] L. Fan, Q. Jacobson, P. Cao, W. Lin. *Web prefetching between low-bandwidth clients and proxies: Potential and performance.* Proc. ACM SIGMETRICS '99, Atlanta, Mai 1999.

[FrBo96a] N. Freed, N. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies.* RFC 2045, IETF, November 1996.

[FrBo96b] N. Freed, N. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types.* RFC 2046, IETF, November 1996.

[GeNi98] J. Gettys, H. F. Nielsen. *SMUX Protocol Specification.* World Wide Consortium, Juli 1998.

[Glas94] S. Glassman. *A Caching Relay for the World Wide Web*. 1. International World Wide Web Conference Conference, Genf, May 1994.

[Goo01] *Google.* http://www.google.com/, Dezember 2001.

[GRC97] S. Gadde, M. Rabinovich, J. Chase, M. Rabinovich. *Reduce, reuse, recycle: an approach to building large Internet caches.* Proc. 6th Workshop on Hot Topics in Operating Systems, Mai 1997.

[Gri98a] C. Grimm. *Was Web-Master und -Nutzer über den Einsatz von Proxies wissen sollten.* In: Internet – von der Idee zum kommerziellen Einsatz. Deutscher Internet Kongress 1998 / Irene Heinen (Hrsg.). S. 198–206. Heidelberg, dpunkt.verlag, 1998.

[Gun96] S. Gundavaram. *CGI Programming on the World Wide Web.* O'Reilly, März 1996. http://www.oreilly.com/openbook/cgi/, Dezember 2001.

[GVP98a] C. Grimm, J.-S. Vöckler, H. Pralle. *Konzeption einer Cache-Server-Infrastruktur auf dem Wissenschaftsnetz.* Abschlussbericht, DFN-Verein, August 1998.

[GVP98b] C. Grimm. J.-S. Vöckler, H. Pralle. *Load and Traffic Balancing in Large Scale Cache Meshes.* Proc. TERENA Networking Conference '98, Dresden, Oktober 1998.

[GVP01] C. Grimm, J.-S. Vöckler, H. Pralle. *Konzeption einer Cache-Server-Infrastruktur auf dem Wissenschaftsnetz – Teil II.* Abschlussbericht, DFN-Verein, Februar 2001.

[GwSe95] J. Gwertzman, M. Seltzer. *The Case for Geographical Push Caching.* Proc. 5th Annual Workshop on Hot Operating Systems, Orcas Island, Mai 1995.

[GwSe96] J. Gwertzman, M. Seltzer. *World Wide Web Cache Consistency.* Proc. USENIX 1996 Annual Technical Conference, San Diego, Januar 1996.

[HaDi97] A. A. Hafez, Y. Ding. *Measuring Web server resource consumption.* Proc. 1997 Computer Measurement Group Conference, Orlando, Dezember 1997.

[HMY97] A. Heddaya, S. Mirdad, D. Yates. *Diffusion based caching along routing paths.* Proc. 2nd Workshop on Caching, Boulder, Juni 1997.

[HOT97] J. Heidemann, K. Obraczka, J. Touch. *Modeling the performance of HTTP over several transport protocols.* IEEE/ACM Transactions on Networking, Vol. 5, No. 5, S. 616–630, Oktober 1997.

[HRW98] M. Hamilton, A. Rousskov, D. Wessels. *Cache Digest specification – version 5.* Dezember 1998.

[IRC02] *IRCache Home.* http://www.ircache.net/. April 2002.

[ISO79] *Standard Generalized Markup Language (SGML).* ISO/IEC 8879, 1979.

[ITG97] *ITG-Empfehlung 5.2-03: Nachrichtenverkehrstheorie.* VDE Verlag, 1997.

[Jai91] R. Jain. *The Art of Computer Systems Performance Analysis.* Wiley & Sons, 1991.

[JKB94] N. L. Johnson, S. Kotz, N. Balakrishnan. *Continuous Univariate Distributions, Volume 1.* Houghton Mifflin, 1994.

[JKK92] N. L. Johnson, S. Kotz, A. W. Kemp. *Univariate Discrete Distributions.* Wiley & Sons, 1992.

[Jon01] M. T. Jones. *An Embeddable HTTP Server.* Dr. Dobb's Journal, Oktober 2001.

[JuCh99] J. Jung, K. Chon. *Replicache: Enhancing Web Caching Architecture with the Replication of Large Object.* Proc. 13th International Conference on Information Networking, Cheju, Januar 1999.

[JVBM98] A. de Jong, T. Verschuren, H. Bekker, I. Melve. *Report on the costs and benefits of operating caching services.* DESIRE – EU Project report D4.2, Januar 1998.

[KBC97] H. Krawczyk, M. Bellare, R. Canetti. *HMAC: Keyed-Hashing for Message Authentication.* RFC 2104, IETF, Februar 1997.

[KCYS99] E. Kawai, K. Chinen, S. Yamaguchi, H. Sunahara. *A Quantitative Analysis of ICP Queries at the Distributed WWW Caching System.* Proc. 9th Annual Conference of the Internet society (INET'99), San Jose, Juni 1999.

[Ken51] D. G. Kendall. *Some problems in the theory of queues.* Journal of the Royal Statistical Society, Series B, Vol. 13, S. 151–173, 1951.

[Ken87] C. A. Kent. *Cache Coherence in Distributed Systems.* Digital Western Research Laboratory, Technical Report 87/4, Dezember 1987. Ebenso: Dissertation Department of Computer Science, Purdue University, August 1986.

[Kle75] L. Kleinrock. *Queueing Systems Volume 1: Theory.* Wiley & Sons,1975.

[KLL+97] D. R. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, R. Panigrahy. *Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web.* Proc. 29th ACM Symposium on Theory of Computing, S. 654–663, Mai 1997.

[KLM97] T. M. Kroeger, D. D. E. Long, J. C. Mogul. *Exploring the Bounds of Web Latency Reduction from Caching and Prefetching.* Proc. 1st USENIX Symposium on Internet Technologies and Systems, Monterey, Dezember 1997.

[KMK99] B. Krishnamurthy, J. C. Mogul, D. M. Kristol. *Key Differences between HTTP/1.0 and HTTP/1.1.* In Proc. 8th WWW Conference, Toronto, Mai 1999.

[Knu81] D. E. Knuth. *The Art of Computer Programming – Vol. 2: Seminumerical Algorithms.* Addison-Wesley, 1981.

[Kob78] H. Kobayashi. *Modeling and Analysis – An Introduction to System Performance Evaluation Methodology.* Addison-Wesley, 1978.

[KrAr99] B. Krishnamurthy, M. Arlitt. *PRO-COW: Protocol Compliance on the Web – A Longitudinal Study.* Proc. 3rd USENIX Symposium on Internet Technologies and Systems, San Francisco, März 2001.

[KrMo00] D. Kristol, L. Montulli. *HTTP State Management Mechanism.* IETF, RFC 2965, Oktober 2000.

[KrRe01] B. Krishnamurty, J. Rexford. *Web Protocols and Practice.* Addison-Wesley, 2001.

[KrWi97] B. Krishnamurthy, C. Wills. *Piggyback Cache Validation for Proxy Caches in the World Wide Web.* Computer Networks and ISDN Systems, Vol. 30, Nr. 1–7, S. 185–193,1998.

[LaGe99] Y. Lafon, J. Gettys. *(Proposed) HTTP-NG Working Group.* World Wide Web Consortium, Dezember 1999.
http://www.w3.org/Protocols/HTTP-NG/, Dezember 2001.

[LaGi99] S. Lawrence, C. L. Giles. *Accessibility of Information on the Web.* Nature, Vol. 400, S. 107–109, Juli 1999.

[LaKe00] A. M. Law, W. D. Kelton. *Simulation Modeling and Analysis.* McGraw-Hill, 2000.

[LCD00] D. Li, P. Cao, M. Dahlin. *Web Cache Invalidation Protocol.* Internet Draft, IETF. November 2000.

[LiCa97] C. Liu, P. Cao. *Maintaining Strong Cache Consistency in the World-Wide Web.* Proc. 17th International Conference on Distributed Computing Systems, Baltimore, Mai 1997.

[LiCh99] D. Li, D. R. Cheriton. *Scalable Web Caching of Frequently Updated Objects using Reliable Multicast.* Proc. 2nd USENIX Symposium on Internet Technologies and Systems, Boulder, Oktober 1999.

[Lip68] J. S. Liptay. *Structural Aspects of the System/360 Model 85 Part II: The Cache.* IBM Systems Journal, Vol. 7, Nr. 1, S. 15–21, Januar 1968.

[Lit61] J. D. C. Little. *A proof for the queueing formula: $L=\lambda w$.* Operations Research, Vol. 9, Nr. 3, S 383–387, Mai 1961.

[Luo94] A. Luotonen. *World-Wide Web Proxies.* 1. International World Wide Web Conference Conference, Genf, Mai 1994.

[Luo97] A. Luotonen. *Web Proxy Servers.* Prentice Hall, 1997.

[Mah99] A. Mahanti. *Web Proxy Workload Characterisation and Modelling.* M. Sc. Thesis, Department of Computer Science, University of Saskatchewan, September 1999.

[MCF00] M. Molina, P. Castelli, G. Foddis. *Web Traffic Modeling Exploiting TCP Connections' Temporal Clustering through HTML-REDUCE.* IEEE Network, Vol. 14, Nr. 3, S. 46–55 , Mai/Juni 2000.

[MDFK97] J. C. Mogul, F. Douglis, A. Feldmann, B. Krishnamurthy. *Potential benefits of delta encoding and data compression for HTTP.* Proc. ACM SIGCOMM '97, Cannes, September 1997.

[MeAl98] D. A. Menascé, V. A. F. Almeida. *Capacity Planning for Web Performance.* Prentice Hall, 1998.

[MEW00] A. Mahanti, D. Eager, C. Williamson. *Temporal Locality and its Impact on Web Proxy Cache Performance.* Performance Evaluation Journal: Special Issue on Internet Performance Modelling, Vol. 42, Nr. 2/3, S. 187–203, September 2000.

[Mil92] D. L. Mills. *Network Time Protocol (Version 3).* RFC 1305, IETF, März 1992.

[Mog95a] J. C. Mogul. *The Case for Persistent-Connection HTTP.* Proc. ACM SIGCOMM '95 Symposium on Communications Architectures and Protocols, Cambridge, August 1995.

[Mog95b] J. C. Mogul. *Operating system support for busy interent servers.* Proc. 5th Workshop on Hot Topics in Operating Systems, Orcas Islands, Mai 1995.

[Mog95c] J. Mogul. *Network behavior of a busy web server and its clients.* Technical Report Research Report 95/5, Digital Equipment Corporation, Oktober 1995.

[Mog99] J. C. Mogul. *Errors in timestamp-based HTTP header.* Compaq Western Research Lab, Research Report 99/3, Dezember 1999.

[MS97] *Cache Array Routing Protocol and Microsoft Proxy Server 2.0.* Microsoft White Paper, Mocrosoft Corporation, 1997.

[MSBV97] I. Melve, L. Slettjord, H. Bekker, T. Verschuren. *Building a Web Caching System – Architectural Considerations.* Proc. 8th Joint European Networking Conference, Edinburgh, Mai 1997.

[MVBS98] I. Melve, T. Verschuren, H. Bekker, L. Slettjord. *Practical Experiences of Establishing Web Proxy Caching Meshes.* DESIRE EU Project report D4.3, Februar 1998.

[NBK02] E. Nahum, T. Barzilai, D. D. Kandlur. *Performance Issues in WWW Servers.* IEEE/ACM Transactions on Networking, Vol. 10, Nr. 1, S. 2–11, Februar 2002.

[NeHe97] J. Z. Neisser, J. Heaton. *Cooperative Web Caching using Multicasting over a Metropolitan Area Network in Greater Manchester.* Proc. 8th Joint European Networking Conference, Edinburgh, Mai 1997.

[Net01] *Network Appliance.* http://www.netapp.com/, Dezember 2001.

[Net02] Netcraft Ltd. *Web Server Survey.* http://www.netcraft.com/survey/, Februar 2002.

[NGBS+97] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. Lie, C. Lilley. *Network Performance Effects of HTTP/1.1, CSS1, and PNG.* Proc. ACM SIGCOMM '97, Cannes, September 1997.

[PaAl00] V. Paxson, M. Allman. *Computing TCP's Retransmission Timer.* IETF, RFC 2988, November 2000.

[PaFl95] V. Paxson, S. Floyd. *Wide area traffic: the failure of Poisson modeling.* IEEE/ACM Transactions on Networking, Vol. 3, Nr. 3, S. 226–244, Juni 1995.

[PaFl97] V. Paxson S. Floyd. *Why we don't know how to simulate the Internet.* Proc. 1997 Winter simulation conference, Atlanta, Dezember 1997.

[PaMo95] V. N. Padmanabhan, J. C. Mogul. *Improving HTTP Latency.* Computer Networks and ISDN Systems, Vol. 28, S. 25–35, Dezember 1995.

[Pav01] *Pavuk – World Wide Web Mirror.* http://www.pavuk.org/, Dezember 2001.

[Pax99] V. Paxson. *End-to-End Internet Packet Dynamics.* IEEE/ACM Transactions on Networking, Vol. 7, Nr. 3, S. 277–292, Juni 1999.

[Pem+00] S. Pemberton et. al. *XHTML™ 1.0: The Extensible HyperText Markup Language.* World Wide Web Consortium, Januar 2000.

[PFTV90] W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling. *Numerical Recipes in C.* Cambridge University Press, 1990.

[Pit98] J. E Pitkow. *Summary of WWW Characterizations.* World Wide Web, Special Issue on Characterization and Performance Evaluation, Oktober 1999.

[PoHa97] D, Povey, J. Harrison. *A Distributed Internet Cache.* Proc. 20th Australian Computer Science Conference, Sydney, Februar 1997.

[Pol01] *Web Polygraph.* http://www.web-polygraph.org/, Dezember 2001.

[Pos81] J. Postel. *Transmission Control Protocol.* STD 7, RFC 793, IETF, September 1981.

[RBR01] P. Rodriguez, E.W. Biersack, K.W. Ross. *Automated Delivery of Web Documents Through a Caching Infrastructure.* Submitted Paper.

[Res00] E. Rescorla. *HTTP Over TLS.* RFC 2818, IETF, Mai 2000.

[Ri92] R. Rivest. *The MD5 Message-Digest Algorithm.* RFC 1321, IETF, April 1992.

[RLHJ99] D. Raggett, A. Le Hors, I. Jacobs. *HTML 4.01 Specification.* World Wide Web Consortium, Dezember 1999.
http://www.w3.org/TR/REC-html40/, Dezember 2001.

[Ros97] K. W. Ross. *Hash-Routing for Collections of Shared Web Caches.* IEEE Network Magazine, Vol. 11, Nr. 6, S. 37–44, November/Dezember 1997.

[RoSo99] A. Rousskov, V. Soloviev. *A performance study of the Squid proxy on HTTP/1.0.* WorldWide Web Journal, Special Edition on WWW Characterization and Performance and Evaluation Vol. 2, Nr. 1/2, S. 47–67, 1999.

[RoWe98] A. Rousskov, D. Wessels. *Cache digests.* Computer Networks and ISDN Systems, Vol. 30, S. 2155–2168, November 1998.

[RoWe00] A. Rousskov, D. Wessels. *The Third Cache-Off.* Oktober 2000. http://www.measurement-factory.com/results/public/cacheoff/N03/, Dezember 2001.

[RRB98] P. Rodriguez, K.W. Ross, E. Biersack. *Distributing Frequently-Changing Documents in the Web: Multicasting or Hierarchical Caching.* Computer Networks and ISDN Systems, Vol. 30, S. 2223–2243, November 1998.

[RSB01] P. Rodriguez, C. Spanner, E. W. Biersack. *Analysis of Web Caching Architectures: Hierarchical and Distributed Caching.* IEEE/ACM Transactions on Networking, Vol. 9, Nr. 4, S. 404–418, August 2001.

[RSGR00] M. S. Raunak, P. Shenoy, P. Goyal, K. Ramamritham. *Implications of Proxy Caching for Provisioning Servers and Networks.* Proc. ACM SIGMETRICS 2000, Santa Clara, Juni 2000.

[RYHE99] R. Rejaie, H. Yu, M. Handley, D. Estrin. *Multimedia Proxy Caching Mechanism for Quality Adaptive Streaming Applications in the Internet.* Proc. Fourth International Web Caching Workshop, San Diego, April 1999.

[Sch00] R. Schlittgen. *Einführung in die Statistik: Analyse und Modellierung von Daten.* 9. Aufl., Oldenbourg, 2000.

[Sed90] R. Sedgewick. *Algorithms in C.* Addison-Wesley, 1990.

[SGHS01] E. Shriver, E. Gabber, L. Huang, C. A. Stein. *Storage management for web proxies.* Proc. 2001 USENIX Annual Technical Conference, Boston, Juni 2001.

[Sha86] M. Shapiro. *Structure and Encapsulation in Distributed Systems: The Proxy Principle.* Proc. 6th International Conference on Distributed Computer Systems, Cambridge, Mai 1986.

[Smi94] N. Smith. *What can Archives offer the World Wide Web.* 1. International World Wide Web Conference Conference, Genf, May 1994.

[SoMa94] K. Sollins, L. Masinter. *Functional Requirements for Uniform Resource Names.* RFC 1737, IETF, Dezember 1994.

[Ste99] W. R. Stevens. *UNIX Network Programming, Vol. 1.* Prentice Hall, 2. Aufl, 1999.

[StSi00] D. Starobinski, M. Sidi. *Modeling and Analysis of Power-Tail Distributions via Classical Teletraffic Methods.* Queueing Systems, Vol. 36, Nr. 1–3, S. 243–267, November 2000.

[TaKa98] H. M. Taylor, S. Karlin. *An Introduction to Stochastic Modeling.* Academic Press, 3rd Edition, 1998.

[TDVK98] R. Tewari, M. Dahlin, H. M. Vin, J. S. Kay. *Design Considerations for Distributed Caching on the Internet.* Proc. 19th International Conference on Distributed Computing and Systems, Austin, Mai 1999.

[TML99] G. Tomlinson, D. Major, R. Lee. *High-Capacity Internet Middleware: Internet Caching System Architectural Overview.* Proc. 2nd Workshop on Internet Server Performance, Atlanta, Mai 1999.

[ToHu98] J. Touch, A. S. Hughes. *LSAM proxy cache: a multicast distributed virtual cache.* Computer Networks and ISDN Systems, Vol. 30, S. 2245–2252, November 1998.

[Tra96] P. Tran-Gia. *Analytische Leistungsbewertung verteilter Systeme.* Springer, 1996.

[Tro02] Trolltech AS. *Qt 3.0.* http://www.trolltech.com/, März 2002.

[VaRo98] V. Valloppillil, K. W. Ross. *Cache Array Routing Protocol v1.0.* Internet Draft (expires August 1998), IETF, Februar 1998.

[VGP98] J.-S. Vöckler, C. Grimm, H. Pralle. *Request routing in cache meshes.* Computer Networks and ISDN Systems, Vol. 30, Nr. 22–23, S. 2269–2278, November 1998.

[ViWe00] P. Vixie, D. Wessels. *Hyper Text Caching Protocol (HTCP/0.0).* RFC 2756, IETF, Januar 2000.

[VJBM98] T. Verschuren, A. de Jong, H. Bekker, I. Melve. *Web Caching Meshes: Hit or Miss?* Proc. 8th Annual Conference of the Internet society, Genf, Juni 1998.

[W3C01] World Wide Web Consortium, http://www.w3.org/Protocols/, Dezember 2001.

[Wan99] J. Wang. *A Survey of Web Caching Schemes for the Internet.* ACM Computer Communication Review , Vol. 29, Nr. 5, S. 36–46, Oktober 1999.

[WeCl97a] D. Wessels, K. Claffy. *Internet Cache Protocol (ICP), version 2.* RFC 2186, IETF, September 1997.

[WeCl97b] D. Wessels, K. Claffy. *Application of Internet Cache Protocol (ICP), version 2.* RFC 2186, IETF, September 1997.

[WeCl98] D. Wessels, K. Claffy. *ICP and the Squid web cache.* IEEE Journal on Selected Areas in Communications, Vol. 16, Nr. 3, S. 345–357, April 1998.

[Wes01a] D. Wessels. *Web Caching.* O'Reilly, Juni 2001.

[Wes01b] D. Wessels. *SQUID Frequently Asked Questions.* http://www.squid-cache.org/Doc/FAQ/FAQ.html, Dezember 2001.

[Wes02] D. Wessels et. al. *NLANR Hierarchical Caching System Usage Statistics.* http://www.ircache.net/Statistics/, Februar 2002.

[Wil65] M. V. Wilkes. *Slave Memories and Dynamic Storage Allocation.* IEEE Transactions on Electronic Computers, Vol. 14, Nr. 2, S. 270–271, April 1965.

[Wil02] C. Williamson. *On Filter Effects in Web Caching Hierarchies.* ACM Transactions on Internet Technology, Vol. 2, Nr. 1, S. 47–77, Februar 2002.

[WiMi99] C. E. Wills, M. Mikhailov. *Examining the Cacheability of User-Requested Web Resources.* Proc. 4th International Web Caching Workshop, San Diego, Juni 1999.

[WVS+99] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, H. M. Levy. *On the scale and performance of cooperative Web proxy caching.* Operating Systems Review Nr. 34, Vol. 5, S. 16–31, Dezember 1999.

[YBS99] H. Yu, L. Breslau, S. Shenker. *A Scalable Web Cache Consistency Architecture.* Proc. ACM SIGCOMM '99, S. 163–174, September 1999.

[ZMN+98] L. Zhang, S. Michel, K. Nguyen, A. Rosenstein, S. Floyd, V. Jacobson. *Adaptive Web Caching: Towards a New Global Caching Architecture.* Computer Networks and ISDN Systems, Vol. 30, S. 2169–2177, November 1998.

Tabellarischer Lebenslauf

Christian Grimm

Persönliche Daten

12. Juli 1966	geboren in Minden als Sohn von Ernst-Ludwig Grimm und seiner Ehefrau Gerda, geb. Oldemeyer.
seit 1995	Lebensgemeinschaft mit Yvonne Scherzer.
Mai 2000	Geburt der Söhne Benno und Lennart.

Schulbesuch und Ausbildung

1972 – 1976	Grundschule Minden und Barkhausen / Porta Westfalica.
1976 – 1985	Ratsgymnasium Minden, Abschluss mit dem Abitur.
1985 – 1986	Grundwehrdienst.
1986 – 1989	Berufsausbildung bei der Firma Schoppe & Faeser in Minden, Abschluss als Informationselektroniker.
1989 – 1995	Universität Hannover, Studium der Elektrotechnik mit dem Schwerpunkt Nachrichtenverarbeitung.
Februar 1995	Abschluss des Studiums mit der Diplomhauptprüfung.

Beruflicher Werdegang

1995 – 1997	Universität Hannover, Lehrgebiet Rechnernetze und Verteilte Systeme. Wissenschaftlicher Mitarbeiter, Betreuung der Vorlesungen „Rechnernetze und Verteilte Systeme“, Mitarbeit im Projekt „Regionales Testbed Nord im DFN“.
1996 – 2001	Universität Hannover, Lehrgebiet Rechnernetze und Verteilte Systeme. Projektleiter für die DFN-Projekte „Konzeption einer Cache-Server-Infrastruktur auf dem Wissenschaftsnetz, Teil 1 und 2“.
seit 1998	Universität Hannover, Regionales Rechenzentrum für Niedersachsen / Lehrgebiet Rechnernetze und Verteilte Systeme. Wissenschaftlicher Assistent, Koordination für den Bereich F&E.