Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik, Institut für Verteilte Systeme

Diplomarbeit

Analyse, Konzeption und Realisierung eines Proxycache-Systems zur Erhöhung der Dienstgüte im WWW

Verfasser:

Jens Elkner

Betreuer:

Prof. Reiner Dumke

Dr. Achim Winkler

01. Februar 1999

Bibliographische Angaben:

Elkner, Jens:
Diplomarbeit: „Analyse, Konzeption und Realisierung eines Proxycache-Systems zur Erhöhung der Dienstgüte im WWW“.
Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik,
Institut für Verteilte Systeme, Februar 1999.
148 Seiten, 61 Abbildungen, 18 Tabellen

Copyright © 1999 by Jens Elkner, Walther-Rathenau-Str. 58, D-39104 Magdeburg.

Alle Rechte vorbehalten. Kein Teil dieser Arbeit darf in irgendeiner Form (Druck, Fotokopie oder einem anderen Verfahren) ohne schriftliche Genehmigung des Autors reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Kurzfassung

Diese Arbeit wird ausgehend von einer immer stärker werdenden Belastung des Internet durch das World Wide Web, Möglichkeiten einer Lastreduzierung diskutieren.

Anschließend wird detailliert auf die Funktionsweise von Proxycaches im allgemeinen und Squid im speziellen eingegangen. Es werden die Möglichkeiten eines Proxycache-Verbundes und der derzeit dafür verfügbaren Kommunikations-Protokolle erläutert, sowie evtl. Probleme und deren Lösungen dargestellt.

Des weiteren wird der Versuch unternommen, die wichtigsten, für einen optimalen Proxycache-Dienst erforderlichen Parameter am Beispiel von Squid aufzuzeigen und Empfehlungen sowohl für die Proxycache-Konfiguration als auch die Administration von WWW-Servern und Webautoren auszusprechen. Last but not least wird die Verbesserung des Proxycache-Dienstes durch den Einsatz von load balancing diskutiert, bereits vorhandene als auch vom Autoren abgewandelte Verfahren analysiert und ausgewertet.

Abstract

Considering the continuous growth of network traffic in the internet caused by the World Wide Web, this thesis discusses the opportunities to reduce network load.

Subsequently a detailed description of the functionality of proxycaches in general and specifically of Squid is given. Opportunities of proxycache meshes and the related available communication protocols are explained as well as the eventual problems and solutions.

Furthermore the thesis discusses the most important parameters for an optimal proxycache service as an example of Squid and giving recommendations for proxycache configurations, for WWW server administration and Web authors. Last but not least possible improvements of the proxycache service by using load balancing are discussed and already known methods as well as methods, modified by the author, are analyzed and evaluated.

Aufgabenstellung

Platzhalter für das Aufgabenstellungsblatt

Inhaltsverzeichnis

1 Einführung und Motivation 15

1.1 Das Internet 15
1.2 Das World Wide Web 16
1.3 Der Internet-Killer WWW 18
1.4 Möglichkeiten zur Verbesserung der Dienstgüte 20

2 Proxycaches 23

2.1 Das Internet Client-Server Modell 23
2.1.1 Was ist ein Proxy ? 24
2.1.2 Was ist ein Proxycache ? 26
2.2 Cache Hierarchien 27
2.2.1 Notwendigkeit eines Proxycache-Verbunds 27
2.2.2 Notwendigkeit einer Proxycache-Hierarchie 28
2.2.3 Cache-Protokolle 33
2.2.3.1 Das Internet Cache Protocol (ICP) 33
2.2.3.2 Das Web Cache Control Protocol (WCCP) 36
2.2.3.3 Das Cache Array Routing Protocol (CARP) 38
2.2.3.4 Das Hyper Text Caching Protocol (HTCP) 40
2.2.3.5 Cache Digests 40
2.3 Squid 42
2.3.1 Was ist Squid 42
2.3.2 Squid an der Universität Magdeburg 46

3 Performance Tuning und Konsistenz von Cache-Objekten 51

3.1 Konfiguration und Anforderungen von Squid 51
3.1.1 Festplatten-Speicher (Cache Area) 52
3.1.2 Hauptspeicher 55
3.1.3 Feinabstimmung des Betriebssystems 58
3.2 Konsistenz von Cache-Objekten 59
3.2.1 Squids Regeln für das Cachen von Objekten 60
3.2.2 Einfluß von WWW-Servern 63
3.2.3 Einfluß von User Agents 65
3.3 Redirector 67
3.4 Monitoring 70
3.4.1 Der Squid-Cache-Manager 70
3.4.2 Andere Monitor-Programme 72
3.4.3 Empfehlungen für zu beobachtende Proxycache-Status-Daten 79

4 Lastenausgleich (load balancing) 83

4.1 Proxy-Konfiguration von Clients 83
4.1.1 Die manuelle Proxy-Konfiguration 83
4.1.2 Automatische Proxy-Konfiguration 86

4.2 IP Network Address Translation (NAT) 89
4.3 Cache-Hierarchie 91
4.3.1 CARP und Squid-CARP 93
4.3.2 Simple Hashes 94
4.3.3 Vorschläge für weitere Tests 96

5 Zusammenfassung und Ausblick 99

Anhang A 103

A.1 Anzahl der im DNS registrierten Maschinen 103
A.2 Regressionsanalyse zu NFSNET Statistiken 103
A.3 Das Hypertext-Transfer-Protokoll 107
A.4 Uniform Resource Locators 108
A.5 User Agent Statistiken 109
A.6 Tests zum Lastenausgleich 111
A.6.1 CARP (Test 1) und Squid-CARP (Test 2) 112

Anhang B 127

B.1 Konfiguration des Proxycache.CS.Uni-Magdeburg.De 127
B.2 Squid Redirector Konfiguration (Jesred 1.2) 134
B.2.1 Jesred Konfigurationsdatei 134
B.2.2 Jesred ACL Konfigurationsdatei 135
B.2.3 Jesred Redirect Rules Konfigurationsdatei 135
B.3 PCMS-Konfigurations-Datei für die MCH 138

Literaturverzeichnis 139

Selbständigkeitserklärung 145

Thesen 147

Abbildungsverzeichnis

Abbildung 1.1: Anzahl der WWW-Server im Internet (Tendenz) 17
Abbildung 1.2: Datenverkehr im NSFNET bzgl. wichtiger Protokolle 18
Abbildung 2.1: Einfache Client-Server-Kommunikation 23
Abbildung 2.2: Proxy-Server als Diensterbringer 25
Abbildung 2.3: Client-Server-Kommunikation über einen Proxycache (MISS) 26
Abbildung 2.4: Client-Server-Kommunikation über einen Proxycache (HIT) 27
Abbildung 2.5: Elemente einer Proxycache-Hierarchie 29
Abbildung 2.6: Verteilung von Proxy-Anfragen an http://ivs.cs.uni-magdeburg.de/ 44
Abbildung 2.7: Verteilung von Proxy-Anfragen an http://www.cs.uni-magdeburg.de/ 45
Abbildung 2.8: Verteilung von Proxy-Anfragen an http://java.cs.uni-magdeburg.de/ 45
Abbildung 2.9: Proxycache-Hierarchie an der Universität Magdeburg im September 1996 47
Abbildung 2.10: cache_host_domain - Aufteilung in der DFN-Cache-Hierarchie 49
Abbildung 3.1: Schematische Darstellung der Cache-Verzeichnis-Struktur. 53
Abbildung 3.2: Flußdiagramm zur Überprüfung der Freshness eines Objekts 61
Abbildung 3.3: Squid Cache Manager: Allgemeine Informationen 71
Abbildung 3.4: PCMS/MRTG Statistiken: LRU Age 75
Abbildung 3.5: 3-Schichten-Modell eines neuen PCMS. 77
Abbildung 4.1: Manuelle Proxy-Konfiguration von Netscape (links) und MSIE (rechts) 84
Abbildung A.1: Datenvolumen wichtiger Protokolle im NSFNET. 106
Abbildung A.2: Trend bzgl. Datenvolumen wichtiger Protokolle im NSFNET 106
Abbildung A.3: User Agent Statistiken von WWW-Servern 110
Abbildung A.4: LB (absolut): CARP (left), Squid-CARP (right); total: 724.091 requests 112
Abbildung A.5: LB (absolut): CARP (left), Squid-CARP (right); total: 813.652 requests 112
Abbildung A.6: LB (absolut): CARP (left), Squid-CARP (right); total: 797.286 requests 112
Abbildung A.7: LB (absolut): CARP (left), Squid-CARP (right); total: 1.012.006 requests 113
Abbildung A.8: LB (absolut): CARP (left), Squid-CARP (right); total: 1.065.412 requests 113
Abbildung A.9: LB (absolut): CARP (left), Squid-CARP (right); total: 1.202.550 requests 113
Abbildung A.10: LB (absolut): CARP (left), Squid-CARP (right); total: 1.083.031 requests 114
Abbildung A.11: LB (absolut): CARP (left), Squid-CARP (right); total: 947.991 requests 114
Abbildung A.12: LB (absolut): CARP (left), Squid-CARP (right); total: 960.227 requests 114
Abbildung A.13: LB (absolut): CARP (left), Squid-CARP (right); total: 1.257.210 requests 115
Abbildung A.14: LB (absolut): CARP (left), Squid-CARP (right); total: 8.222.591.572 bytes. 115
Abbildung A.15: LB (absolut): CARP (left), Squid-CARP (right); total: 8.720.394.922 bytes. 115
Abbildung A.16: LB (absolut): CARP (left), Squid-CARP (right); total: 10.801.065.979 bytes. . . .116
Abbildung A.17: LB (absolut): CARP (left), Squid-CARP (right); total: 12.688.800.455 bytes. . . .116
Abbildung A.18: LB (absolut): CARP (left), Squid-CARP (right); total: 12.427.657.633 bytes. . . .116
Abbildung A.19: LB (absolut): CARP (left), Squid-CARP (right); total: 11.719.732.474 bytes. . . .117
Abbildung A.20: LB (absolut): CARP (left), Squid-CARP (right); total: 11.630.194.766 bytes. . . .117
Abbildung A.21: LB (absolut): CARP (left), Squid-CARP (right); total: 14.309.593.596 bytes. . . .117
Abbildung A.22: LB (absolut): CARP (left), Squid-CARP (right); total: 11.359.838.505 bytes. . . .118
Abbildung A.23: LB (absolut): CARP (left), Squid-CARP (right); total: 13.249.527.918 bytes. . . .118
Abbildung A.24: LB (relativ): CARP (left), Squid-CARP (right); total: 724.091 requests 118
Abbildung A.25: LB (relativ): CARP (left), Squid-CARP (right); total: 813.652 requests 119
Abbildung A.26: LB (relativ): CARP (left), Squid-CARP (right); total: 797.286 requests 119
Abbildung A.27: LB (relativ): CARP (left), Squid-CARP (right); total: 1.012.006 requests 119
Abbildung A.28: LB (relativ): CARP (left), Squid-CARP (right); total: 1.065.412 requests 120

Abbildung A.29: LB (relativ): CARP (left), Squid-CARP (right); total: 1.202.550 requests 120
Abbildung A.30: LB (relativ): CARP (left), Squid-CARP (right); total: 1.083.031 requests 120
Abbildung A.31: LB (relativ): CARP (left), Squid-CARP (right); total: 947.991 requests 121
Abbildung A.32: LB (relativ): CARP (left), Squid-CARP (right); total: 960.227 requests 121
Abbildung A.33: LB (relativ): CARP (left), Squid-CARP (right); total: 1.257.210 requests 121
Abbildung A.34: LB (relativ): CARP (left), Squid-CARP (right); total: 8.222.591.572 bytes 122
Abbildung A.35: LB (relativ): CARP (left), Squid-CARP (right); total: 8.720.394.922 bytes 122
Abbildung A.36: LB (relativ): CARP (left), Squid-CARP (right); total: 10.801.065.979 bytes 122
Abbildung A.37: LB (relativ): CARP (left), Squid-CARP (right); total: 12.688.800.455 bytes 123
Abbildung A.38: LB (relativ): CARP (left), Squid-CARP (right); total:12.427.657.633 bytes. 123
Abbildung A.39: LB (relativ): CARP (left), Squid-CARP (right); total: 11.719.732.474 bytes 123
Abbildung A.40: LB (relativ): CARP (left), Squid-CARP (right); total: 11.630.194.766 bytes 124
Abbildung A.41: LB (relativ): CARP (left), Squid-CARP (right); total: 14.309.593.596 bytes 124
Abbildung A.42: LB (relativ): CARP (left), Squid-CARP (right); total: 11.359.838.505 bytes 124
Abbildung A.43: LB (relativ): CARP (left), Squid-CARP (right); total:13.249.527.918 bytes. 125

Tabellenverzeichnis

Tabelle 2.1: Auszug aus der derzeitigen DFN-Cache-Hierarchie 31
Tabelle 2.2: ODER Verknüpfung von Cache-Set-Matrizen: Wertetabelle 32
Tabelle 2.3: Prozentsätze von Anfragen universitätsinterner Proxycaches 44
Tabelle 2.4: Beziehungen von und zu Proxycaches in der MCH 50
Tabelle 3.1: Aufrüstung und Cache-Verzeichnis-Größen von proxycache.cs 55
Tabelle 3.2: Parameter für die Überprüfung der Freshness eine Objekts 60
Tabelle 3.3: Komponenten des PCMS 72
Tabelle 3.4: Mittels PCMS gesammelte Daten je Proxycache der MCH 80
Tabelle 4.1: Zuteilung der Proxycaches der Universität Magdeburg 85
Tabelle 4.2: Parameter für load balancing Test: CARP 93
Tabelle 4.3: Simple Hashing: Initialisierung der Vergleichswerte 94
Tabelle 4.4: Simple Hash Algorithmen zum Lastenausgleich 95
Tabelle A.1: Anzahl der im DNS registrierten Maschinen 103
Tabelle A.2: Regressionsanalysedaten zum Datenverkehr im NSFNET 105
Tabelle A.3: Aufbau einer Client Anfrage 107
Tabelle A.4: Aufbau einer Antwort eines HTTP-Servers 108
Tabelle A.5: Verschiedene Formen von URL‘s 109
Tabelle A.6: LB CARP, Squid-CARP: Parameter 112

Abkürzungsverzeichnis

ACL	Access Control List
AMS	American Mathematical Society
ATM	Asynchronous Transfer Mode
B-WiN	Breitband-Wissenschaftsnetz des DFN
CARP	Cache Array Routing Protocol
CERN	European Particle Physics Laboratory
CPU	Central Processing Unit
DFN	Verein zur Förderung eines deutschen Forschungsnetzes e.V.
DKRZ	Deutsche Klimarechenzentrum
DoS	Deny of Service
E-Mail	Electronic Mail
FIN	Fakultät für Informatik
FMA	Fakultät für Mathematik
FQDN	Fully Qualified Domain Name
FSU	Friedrich-Schiller-Universität
FTP	File Transfer Protocol
HP	Hewlett Packard
HTCP	Hypertext Caching Protocol
HTTP	Hypertext Transfer Protocol
ICP	Internet Cache Protocol
IRB	Institut für Rechnernetze und Betriebssysteme
IRC	Internet Relay Chat
ISP	Internet Service Provider
IVS	Institut für Verteilte Systeme
LAN	Local Area Network
LRU	Least Recently Used

MCH	Magdeburger Cache-Hierarchie
MIME	Multipurpose Internet Mail Extension
MRZ	Medizinisches Rechenzentrum
MTU	path Maximum Transmission Unit
NASA	National Aeronautics and Space Administration
NIS	Network Information Service
NNTP	Network News Transport Protocol
PAMT	Proxy Array Membership Table
QoS	Quality of Service
RAM	Random Access Memory
RFC	Request For Comments
RTT	Round Trip Time
SCM	Squid Cache Manager
SLD	Second Level Domain
SMTP	Simple Mail Transfer Protocol
TLD	Top Level Domain
TU	Technische Universität
URL	Uniform Resource Locator
URN	Uniform Resource Name
URZ	Universitätsrechenzentrum
WAN	Wide Area Network
WCCP	Web Cache Control Protocol
WiN	Wissenschaftsnetz des DFN
WWW	World Wide Web

1 Einführung und Motivation

1.1 Das Internet

Seit Mitte der 90er Jahre erlebt das Internet den wohl größten Boom seit seinem Bestehen (siehe Anhang A.1und A.2). Nutzten anfänglich hauptsächlich nur Universitäten und Forschungsstätten das Internet, so machen jetzt auch immer mehr Unternehmen sowie private Nutzer von diesem Medium zur Präsentation, Gewinnung und Darstellung verschiedenster Informationen, zu der Bereitstellung von Daten, sowie zur kommerziellen Vermarktung von Produkten gebrauch.

Ursachen für das erst ab ca. 1990 einsetzende, starke Wachstum sind darin zu suchen, daß bis dahin die wichtigsten Mittel zum Austausch von Informationen über das Internet die Dienste E-Mail (Electronic Mail), FTP (File Transfer Protocol) und Usenet (News) waren. All die dafür zur Verfügung stehenden Werkzeuge (Tools) waren relativ kompliziert, teilweise sehr kryptisch zu bedienen und im Grunde genommen nur auf UNIX[1]- und Mainframe-Plattformen verfügbar. Ebenso war es oftmals sehr schwierig, den Ort herauszufinden, an dem sich die benötigten Informationen bzw. Daten befanden, um diese mit den oben genannten Werkzeugen zu beschaffen.

Man erkannte, daß es einen immer größeren Informationsbedarf gab, diesen aber aufgrund der oben genannten Gründe nicht oder nur teilweise befriedigen konnte. Daher wurde 1991 vom Microcomputer Center at the University of Minnesota ein neues Protokoll „Internet Gopher“ [2] erarbeitet, mit dem Zweck, eine verteilte Dokumentensuche und -beschaffung zu ermöglichen. Es ist ein Client-Server-Modell, gekennzeichnet durch eine Hierarchie von Menüs und Verzeichnissen ähnlich einem Filesystem, in dem auch Verknüpfungen zu anderen Gopher-Servern und deren Inhalte möglich sind. Dokumente können damit ohne spezielle Kenntnisse z.B. von FTP auf einfache Art und Weise heruntergeladen werden und auch die Suche nach Informationen wurde damit erleichtert. Dies war schon ein bedeutender Fortschritt in der Geschichte

1. UNIX ist ein Trademark der AT&T Bell Laboratories.

des Internets, da jetzt viele Organisationen in der Lage waren, Informationen im Internet ohne großen Aufwand zur Verfügung zu stellen, die auch von anderen Internet-Teilnehmern gefunden werden können und ebenso Verweise auf bestimmte Wissengebiete direkt in ihrem Server zu integrieren.

1.2 Das World Wide Web

Doch erst das World Wide Web (WWW) machte das Internet auch attraktiv für kommerzielle und private Nutzer. Fast zeitgleich mit dem Gopher System wurde am European Particle Physics Laboratory (CERN) ein Client-Server-Modell entwickelt, um effektiv Informationen und Ideen innerhalb der Organisation auf der Basis von Hypertext austauschen zu können. 1991 / 92 wurde das System so erweitert, daß nun auch das Einbinden von Grafiken sowie die Nutzung von unterschiedlichen Schriftarten und -größen möglich war. Dies und die Entwicklung von Clients mit grafischer Benutzeroberfläche (auch Browser genannt) wie z.B. Mosaic oder Netscape bewegten viele Nutzer dazu, das WWW dem Gopher vorzuziehen, da es noch einfacher zu bedienen war, andere Protokolle wie FTP, Gopher, WAIS ebenfalls unterstützte und eine ansprechende, aber relativ einfache Aufbereitung der publizierten Informationen ermöglichte. Ebenso konnte jetzt i.d.R. jeder Nutzer ohne Hilfe von System-Administratoren eigene Dokumente transparent organisatorisch und inhaltlich im WWW Einordnen und veröffentlichen.

WWW-Browser und -Server sind mittlerweile auf allen modernen Betriebssystemen verfügbar und können somit erstmals relativ einfach auch von privaten Nutzern von zuhause via Modem intensiv genutzt werden. Angesichts immer besserer Tarife der Internet Service Provider (ISP) droht auch die letzte, besonders in Deutschland hohe Hemmschwelle bzgl. der Nutzung des Internet bei privaten Anwendern zu fallen und motiviert, sich via WWW neueste Informationen zu besorgen, Dienstleistungen in Anspruch zu nehmen oder sich einfach mit diesem modernen Medium auf verschiedenste Art und Weise die Zeit zu vertreiben.

Auch erkennen immer mehr Unternehmen das riesige Potential des WWW und nutzen es zur Repräsentation ihrer selbst als auch zur Bereitstellung aktuellster Informationen und Dienstleistungen verschiedenster Art (Online-Banking, Shopping, Software-Upgrades, Unterhaltung, Werbung etc.). Eine wesentliche Rolle hierbei spielt sicher auch die Einsparung erheblicher Kosten für die Erzeugung von Printmedien

und deren Versand oder beispielsweise die Miete für Büros. Die Beobachtung von gängigen Medien wie z.B. Hörfunk und Fernsehen sowie Tagespresse und Magazinen zeigt, daß es inzwischen schon fast zum guten Ruf eines Unternehmens gehört, im WWW präsent zu sein.

Abbildung 1.1 scheint dies zu bestätigen. Zur Erlangung der Daten wurde ein Programm erstellt, welches aus allen bisherigen „Internet Domain Surveys“ [3] die Anzahl aller Rechner mit der Zeichenkette „www“ und „proxy“ bzw. „cache“ im Hostnamen extrahierte. Diese Statistik kann natürlich nur eine Tendenz wiedergeben, da es mit Sicherheit bedeutend mehr WWW-Server gibt, die jedoch nicht die oben angeführten Zeichenketten in ihrem Hostnamen enthalten oder zweitens einen Alias für die entsprechenden WWW-Adresse verwenden. Da im „Internet Domain Survey“ Hostname-Aliase nicht erfaßt werden [4], geht eine vermutlich sehr hohe Anzahl von WWW-Servern und Proxy[cache]s nicht mit in diese Statistik ein.

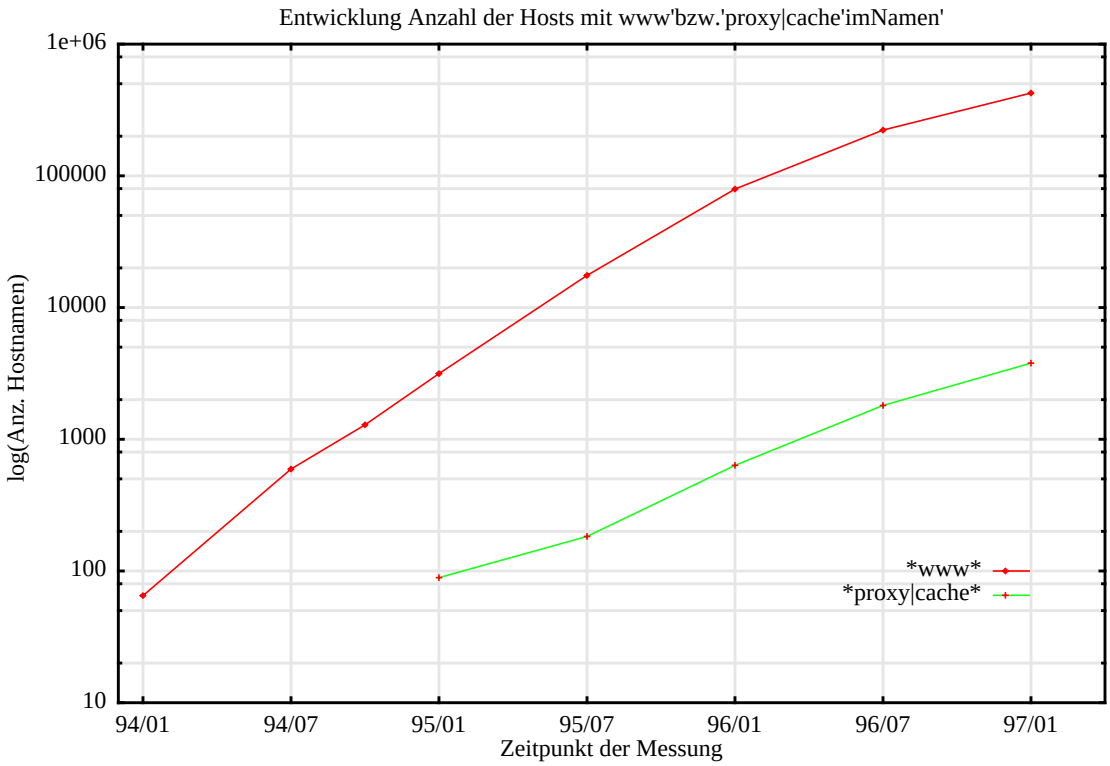

Abbildung 1.1: Anzahl der WWW-Server im Internet (Tendenz)

1.3 Der Internet-Killer WWW

Der exponentielle Anstieg von WWW-Servern (siehe Abbildung 1.1) und der darauf enthaltenen Informationen / Dienste bewirkt auch ein starkes Ansteigen des Verkehrs im Internet. So zeigt Abbildung 1.2 den Zuwachs an transferierten Bytes bzgl. verschiedener Protokolle im NSFNET[1] [5]. Eine genauere Analyse der Daten (siehe Anhang A.2) ergibt, daß der Verkehr gängiger Protokolle wie ftp, gopher, nntp, smtp, telnet, irc im Zeitraum November 1993 bis einschließlich November 1994 zwischen ca. 4% bis 8 % pro Monat gestiegen ist, der Verkehr via WWW[2] jedoch um ca. 25%!

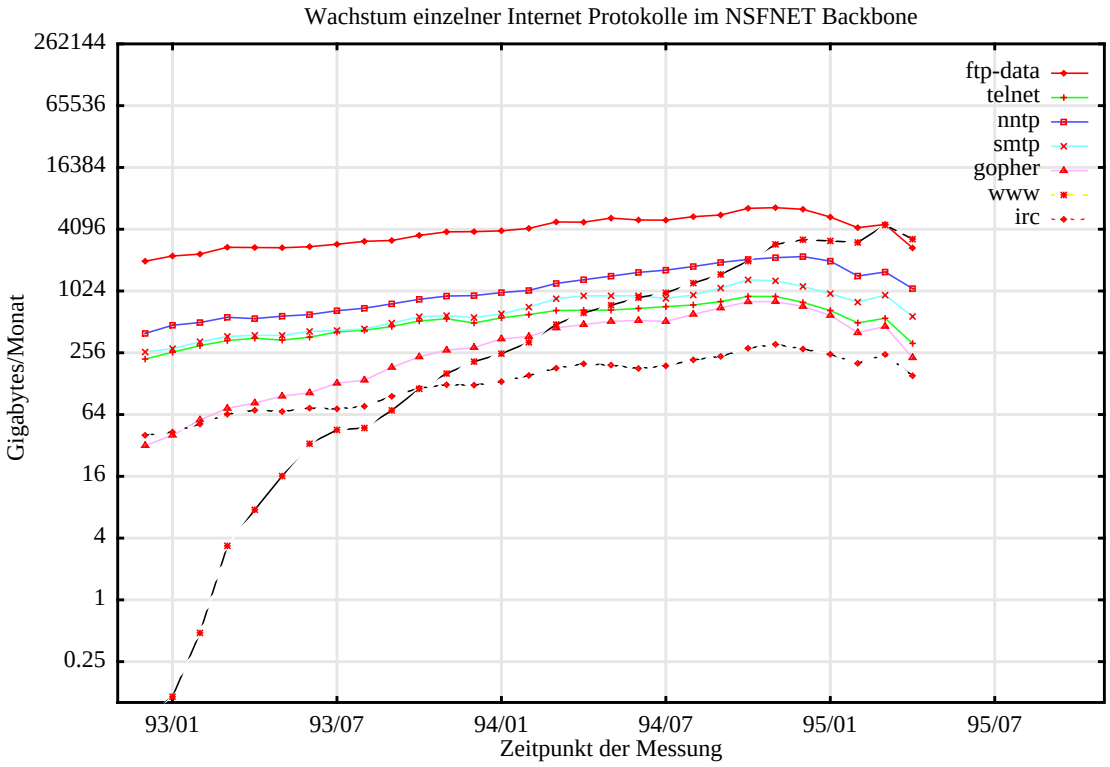

Abbildung 1.2: Datenverkehr im NSFNET bzgl. wichtiger Protokolle

Leider gibt es keine entsprechende, öffentlich verfügbaren Statistiken von anderen Netzen, wie z.B. dem WiN bzw. B-WiN, EuNet oder X-Link, mit denen man ähnliche Analysen bzgl. Netzverkehrstendenz durchführen könnte. Aufgrund eigener Erfahrungen darf jedoch angenommen werden, daß in den Netzen anderer ISP durchaus

1. Seit 30. April 1995 existiert dieses Netz nicht mehr, da es bis zu diesem Zeitpunkt vollständig in die sogenannte NAP Architektur überführt worden war. Das Ende des NSFNET stellte ebenso das Ende der Sammlung der zur Analyse verwendeten Daten dar.
2. Genauer bezieht sich hier WWW auf das Protokoll http und den TCP Port 80.

ähnliche Tendenzen vorherrschen und auch dort inzwischen der durch das WWW verursachte Datenverkehr den aller anderen Protokolle weit übersteigt.

Da immer mehr multimediale Objekte wie Applets, Audio und Video etc. in WWW-Seiten verwendet werden (an dessen Einbindung in den HTML 3.2 Standard bereits seit April ‘96 im Rahmen des W3 Consortium (W3C) gearbeitet [1] wird), geht dies einher mit einer weiteren Erhöhung der Last auf alle involvierten Ressourcen[1]. Stellvertretend für stärker belastete Ressourcen seien hier in erster Linie die zur Verfügung stehende Bandbreite, die Last auf Router, Gateways und Firewalls, die Last auf den WWW-Servern selbst bzgl. Hard- und Software sowie die entstehenden Kosten bzgl. konsumierter Zeit und Bandbreite genannt.

In vielen Fällen resultieren die oben aufgeführten Gründe im allgemeinen in einer geringeren Qualität der meisten Dienste (engl. Quality of Service - QoS) im Internet, was sich z.B. durch lange Ladezeiten von WWW-Objekten oder sogar durch eine Blockierung eines oder mehrerer Server bzw. deren Dienste bemerkbar macht. Oftmals ist dies besonders oft bei FTP-Servern zu beobachten, welche die Anzahl der gleichzeitig angemeldeten anonymen Nutzer oftmals stark begrenzen, um z.B. eigenen lokalen Nutzern die ständige Verfügbarkeit zu garantieren. Auch bei WWW-Servern, die z.B. sehr aktuelle Informationen zu gegebenen Anlässen liefern oder Suchdienste (z.B. Excite, Altavista, Lycos) anbieten und deshalb stark frequentiert sind, tritt der oben genannte Sachverhalt auf. Der diesbezüglich wohl spektakulärste Fall ereignete sich im Juli 1994: verursacht durch die Kollision des Planeten Jupiter mit den Shoemaker-Levy 9 Kometen. Kurz nach der Aufnahme der Bilder mittels in der ganzen Welt verteilter Teleskope wurden diese auf den WWW- und FTP-Servern der NASA veröffentlicht (man sprach in diesem Zusammenhang gelegentlich von „real-time publishing“). Die Serverlast der NASA-Maschinen war so hoch, daß zusätzliche Server aufgesetzt und die gesamten Dokumente zur Reduzierung unnötigen Suchens durch die Clients überarbeitet werden mußten [6]. Innerhalb von 10 Tagen wurden vom WWW-Server des Goddard Space Flight Center 420.000 Zugriffe (Spitzen mit fast 6.000 Zugriffen / Stunde) mit einem gesamten Datentransfervolumen von über 6 Gigabyte registriert.

1. multimediale Objekte sind i.d.R. relativ groß (im Bereich von wenigen KB bis zu mehreren MB)

Aufgrund der schlechten Qualität des WWW, welches hauptsächlich durch langsame, dem Ansturm von WWW-Anfragen nicht gewachsener Hard- und Software sowie zu geringen Bandbreiten zum Herunterladen von entsprechenden Daten bedingt war und heute oftmals noch ist, bildete sich ein zweites Synonym für die Abkürzung WWW heraus: World Wide Wait. Jetzt standen zwar relativ einfach handhabbare Werkzeuge zur Beschaffung von Dokumenten und Daten zur Verfügung, doch das Internet ist der massenhaften Abfrage und dem Transport selbiger nicht oder nur teilweise gewachsen.

All die oben erwähnten Fakten und Beispiele veranlassen immer mehr ISP, die Qualität ihrer Dienste durch den Einsatz geeigneter Mittel, die im folgenden Abschnitt angeführt werden, zu verbessern und durch das Bereitstellen entsprechender Dienste die resultierenden Kosten zu mindern.

1.4 Möglichkeiten zur Verbesserung der Dienstgüte

Eine Möglichkeit zur Verbesserung der Dienstgüte kann die Erhöhung der zur Verfügung stehenden Bandbreite sein. Da dies aber oftmals zu erheblichen Umstrukturierungen in der Netzwerk-Topologie inklusive darin integrierter Komponenten (z.B. Router, Switches, Hubs) sowie zu erheblichen Kosten bzgl. Kauf neuer Netzwerk-Komponenten (z.B. Switches, Router, Kabel) führt und insgesamt nicht unendlich viel Bandbreite zur Verfügung steht, kommt dies für viele ISP relativ selten als wirkliche Möglichkeit in Frage.

Weiterhin werden mit einer verbesserten Bandbreite zwar theoretisch höhere Transferraten möglich, doch wenn der Server überlastet ist, bewirkt dies letztendlich keine oder nur eine sehr geringe Verbesserung der Dienstqualität.

Die derzeit wohl am meisten praktizierte und relativ kostengünstige Methode bzgl. der Anschaffung von Hardware[1] ist die Replikation und das Spiegeln von Daten. So duplizieren beispielsweise sogenannte FTP-Mirrors ganze Filesysteme von Originalservern. Daß dies jedoch auch keine ideale Lösung darstellt, zeigen folgende Sachverhalte:

1. Festplatten werden preislich immer günstiger hinsichtlich Kapazität und Leistung

(1) Oftmals wissen Nutzer gar nicht, daß solche Mirrors in ihrer topologisch gesehenen Nähe existieren, so daß die benötigten Daten vom Originalserver heruntergeladen werden und unwissentlich mit unter eine schlechtere Dienstqualität (lange Ladezeiten) in Kauf genommen werden muß.

(2) Nutzer wissen zwar, daß es einen Spiegel-Server gibt, haben dessen Namen vergessen und können somit nicht die durch die Benutzung des Mirrors entstehenden Vorteile nutzen.

(3) Nutzer trauen dem Mirror nicht und holen sich „sicherheitshalber“ die „neuesten“ Dateien vom Originalserver, weil sie bereits die Erfahrung machen mußten, daß die Daten auf dem Mirror nicht genau denen auf dem Originalserver entsprechen (Grund hierfür wiederum könnte sein, daß die Überprüfung der Konsistenz des Mirrors mit dem Originalserver zu selten oder gar nicht erfolgt bzw. Daten aufgrund fehlender Plattenkapazitäten nur teilweise gespiegelt werden).

(4) Oftmals ist es gar nicht notwendig, gesamte Filesysteme oder Verzeichnisse zu spiegeln, da nur einige wenige Dateien wirklich mehrfach benötigt werden. Somit wird Bandbreite durch das evtl. unnötige Spiegeln und Plattenplatz sowie Prozessorzeit auf dem Server verschwendet.

(5) Aufgrund der Vielfalt der Daten auf den Servern des Internet ist es für einen ISP nahezu unmöglich, alle Daten auf seinem Server zu duplizieren und mit dem Original konsistent zu halten. Ebenso kann der ISP nur mit sehr hohem Aufwand (z.B. Portmonitoring) feststellen, bei welchen Objekten sich eine Duplikation lohnt. Selbst wenn ihm eine entsprechende Liste zur Verfügung steht, stellt sich die Frage nach der Repräsentation dieser Daten. Befinden sich z.B. nur die Hälfte der Dateien eines Verzeichnisses vom Originalserver in dem vom ISP angebotenen Verzeichnis, führt das mit Sicherheit zu dem in (3) genannten Problem...

Um die oben genannten, im Zusammenhang mit Spiegel-Servern auftretenden Probleme zu lösen, werden seit ca. zwei Jahren Untersuchungen zum effektiven Zwischenspeichern von Internet-Objekten (Caching) durchgeführt und entsprechende Soft- und Hardware entwickelt. Wird ein solcher Cache genutzt und werden Objekte mehrmals angefordert, werden diese aus dem gegenüber dem Internet i.d.R. schnelleren Zwischenspeicher (z.B. RAM oder Festplatte) geladen. Die Vorteile sind klar: der Originalserver muß nicht noch einmal kontaktiert werden, womit der Originalserver und Netzwerke wie z.B. WANs entlastet werden und der Client kann die entsprechenden Daten i.d.R. viel schneller laden. Um ein solches Zwischenspeichern möglichst effektiv zu gestalten, stellt sich die Frage, wo man den Zwischenspeicher ansiedelt. Derzeit werden diesbezüglich zwei Arten favorisiert: erstens das Clientcaching und zweitens das Proxycaching. Beim Clientcaching werden die Daten direkt vom Client

lokal zwischengespeichert. Beispiele dafür sind moderne WWW-Browser wie Netscape Navigator, Microsoft Internet Explorer oder Mosaic. Vorteil: die Clients können sehr schnell auf diese Daten zugreifen. Nachteil: Oftmals steht dem Client nur begrenzter Speicherplatz zur Verfügung und andere Clients können diese Daten nicht nutzen. Somit werden Daten immer noch mehrfach von Servern geholt, sobald diese von unterschiedlichen Clients benötigt werden und somit das Netz und die entsprechenden Server belastet. Um auch diese Nachteile zu beseitigen, werden immer häufiger Proxycaches (siehe Abbildung 1.1) eingesetzt, deren Funktionsweise im folgenden Kapitel erläutert wird.

2 Proxycaches

2.1 Das Internet Client-Server Modell

Fast alle Netzwerk-Applikationen sind als Client-Server-Applikation konzipiert bzw. basieren auf Diensten, welche auf dem Client-Server-Modell beruhen. Die Aufgabe der Server besteht darin, einen speziellen Dienst für einen oder mehrere Clients zur Verfügung zu stellen. Eine einfache Client-Server-Kommunikation könnte wie indem in Abbildung 2.1 dargestellten Sequenzdiagramm aussehen: Der Client stellt eine Anfrage an den Server, der Server verbraucht eine gewisse Zeit, um die Anfrage zu verarbeiten und schickt anschließend das Ergebnis der Anfrage an den Client zurück.

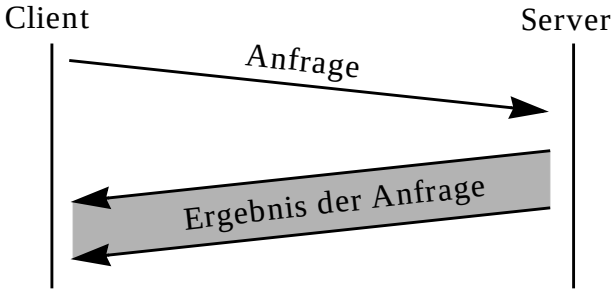

Abbildung 2.1: Einfache Client-Server-Kommunikation

Generell kann zwischen sequentiellen (singlethreaded) und nebenläufigen (multithreaded) Servern unterschieden werden. Ein sequentieller Server wiederholt immer wieder folgende Schritte:

(1) Warte, bis eine Anfrage (request) von einem Client eintrifft
(2) Verarbeite die Anfrage des Client
(3) Sende das Ergebnis der Anfrage an den Client zurück, der diese gestellt hat
(4) Gehe zurück zu Schritt (1)

Das Problem bei sequentiellen Servern ist, daß immer nur eine Anfrage eines Clients verarbeitet und beantwortet werden kann und deshalb entsprechende Wartezeiten be-

züglich der Diensterbringung für Anfragen anderer Clients entstehen. Dieses Problem versucht man mit den nebenläufigen Servern zu beheben, welche folgende Schritte ausführen:

(1) Warte, bis eine Anfrage (request) von einem Client eintrifft
(2) Starte einen Prozeß bzw. Task oder Thread (abhängig vom darunterliegenden Betriebssystem), welcher wiederum folgende Schritte ausführt:
    (a) Verarbeite die Anfrage des Clients
    (b) Sende das Ergebnis der Anfrage an den Client zurück, der diese gestellt hat
    (c) beende dich
(3) Gehe zurück zu Schritt (1)

Der Vorteil der nebenläufigen Server ist, das einfach ein weiterer Server erzeugt wird, der die Anfrage des Clients verarbeitet. Somit werden die Anfragen aller Clients zumindest pseudo-parallel/gleichzeitig bedient. Nachteil dieses Konzeptes ist, daß durch die Erzeugung des neuen Threads, Tasks oder Prozesses weitere Systemressourcen wie z.B. Speicher und besonders CPU-Zeit verbraucht werden und bei sehr vielen gleichzeitigen Anfragen die Performance des Servers negativ beeinflußt wird. Das ist speziell der Fall, wenn schwergewichtige Prozesse erzeugt werden, was z.B. bei der am meisten eingesetzten WWW-Server-Software Apache der Fall ist[1].

Deshalb werden bei modernen Client-Server-Applikationen oftmals schon beim Start des Servers eine bestimmt Anzahl neuer stellvertretender Server erzeugt, welche die Anfragen der Clients bearbeiten und nach Rücksendung des Ergebnisses nicht terminieren und somit eine Art „Server-Cache“ besteht. Erst wenn diese Anzahl an laufenden Server-Prozessen nicht mehr ausreicht, alle Client-Anfragen zu bearbeiten, werden weitere Server-Prozesse entsprechend dem oben angegeben Abarbeitungsschema für nebenläufige Server erzeugt.

2.1.1 Was ist ein Proxy ?

Ein Proxy ist eine Art Stellvertreter (i.d.R. als Software implementiert), welcher Anfragen von gewissen Clients entgegen nimmt und an das beabsichtigte Ziel weiterleitet. Das Ergebnis der Anfrage wird vom Ziel an den Proxy übermittelt, welcher seinerseits dieses dem Client zurückliefern kann.

1. siehe http://www.apache.org/ bzw. http://www.netcraft.com/Survey/

Typischerweise werden Proxies in Firewalls eingesetzt, um die Clients innerhalb eines Firewalls z. B. vor unberechtigten Zugriffen bzw. Sabotage durch sogenannte „Deny of service attacks“ (DoS attacks) von außen zu schützen. D.h. alle Clients innerhalb eines Firewalls können externe Ziele nur über den Proxy und umgekehrt erreichen. Ein Proxy wartet dazu auf die Anfrage eines Clients innerhalb des Firewall, sendet diese weiter an die entsprechenden entfernten (remote) Server außerhalb des Firewall, empfängt die Ergebnisse / Objekte der Anfrage (z.B. ein HTTP-Dokument, eine Datei einer FTP-Anfrage etc.), unterzieht diese laut seinen Regeln einer Prüfung und sendet das Ergebnis letztendlich an den Client zurück (siehe Abbildung 2.2).

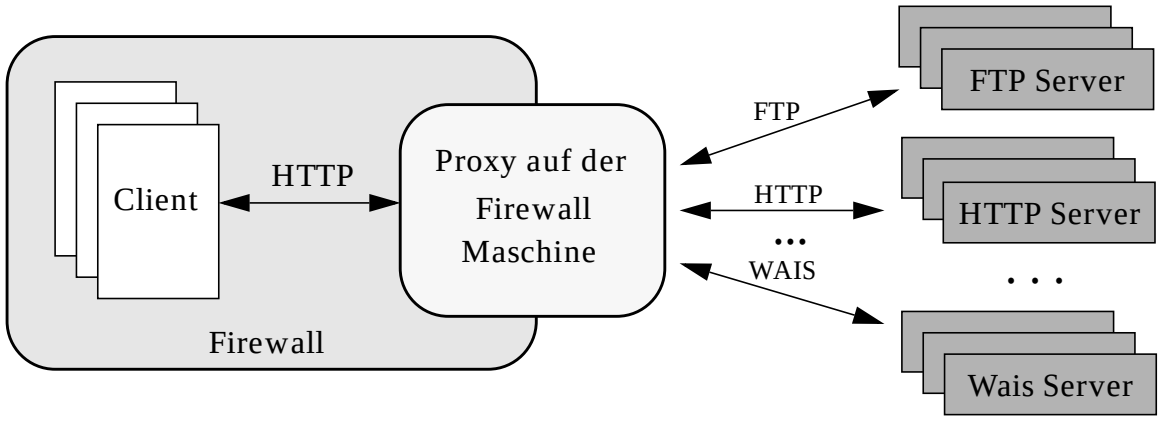

Abbildung 2.2: Proxy-Server als Diensterbringer

Natürlich ist die Nutzung eines Proxies nicht an einen Firewall gebunden, denn auch in anderen Bereichen ist deren Einsatz durchaus sinnvoll (z.B. für Maschinen eines Intranets, die keinen direkten Internet-Anschluß haben oder auch Applikationen, wie z.B. SNMP-Proxies zur Vereinfachung und effektiveren Gestaltung des Netzwerk-Managements).

Wird die obige allgemeine Funktionsweise von Proxies näher betrachtet, fällt auf, daß der Proxy bei *jeder* Client-Anfrage den gewünschten entfernten Server kontaktiert und seinen Regeln entsprechend das Ergebnis an den Client zurückliefert. Wenn kurz nach einer Anfrage ein Client dieselbe Anfrage noch einmal an den Proxy schickt, wird erneut der Server kontaktiert. Damit bekommt der Client zwar die aktuellen Objekte der Anfrage, doch werden i.d.R. unnötig Ressourcen des Proxies und des Servers in Anspruch genommen.

2.1.2 Was ist ein Proxycache ?

Auf der Suche nach effizienten Mechanismen zum verbesserten, schnelleren und ressourcensparenden[1] Zugriff auf Informationen im Internet wurde festgestellt, das gerade Proxies eine Menge Potential bieten, derartige Optimierungen zu erreichen. Man ist hier zu dem Schluß gekommen, daß ein „optimierter“ Proxy nur gewisse von ihm bereits geholte Objekte zwischenspeichern, verwalten und bei gleich lautenden Client-Anfragen selbige an den Client zurücksenden muß, um die obengenannten Ziele zumindest teilweise erreichen zu können.

Dem allgemeinen Prinzip nach ist ein Proxycache also nichts anderes als ein Proxy mit der Besonderheit, daß dieser bestimmte Ergebnisse / Objekte von Client-Anfragen zwischenspeichert (z.B.auf der Festplatte) und bei wiederholter Anfrage eines Clients sofort das zwischengespeicherte Objekt zurückliefert, wenn seine Regeln dies zulassen.

Abbildung 2.3 verdeutlicht die Client-Server-Kommunikation über einen Proxycache, wenn sich das vom Client angeforderte Objekt noch nicht im Cache befindet (MISS). Hier muß, wie bei einem normalen Proxy auch, eine Verbindung zum Server aufgebaut und die Anfrage des Clients weitergereicht werden. Der Proxycache empfängt dann das Ergebnis dieser Anfrage, speichert dieses in seinem Cache ab und sendet es weiter an den Client.

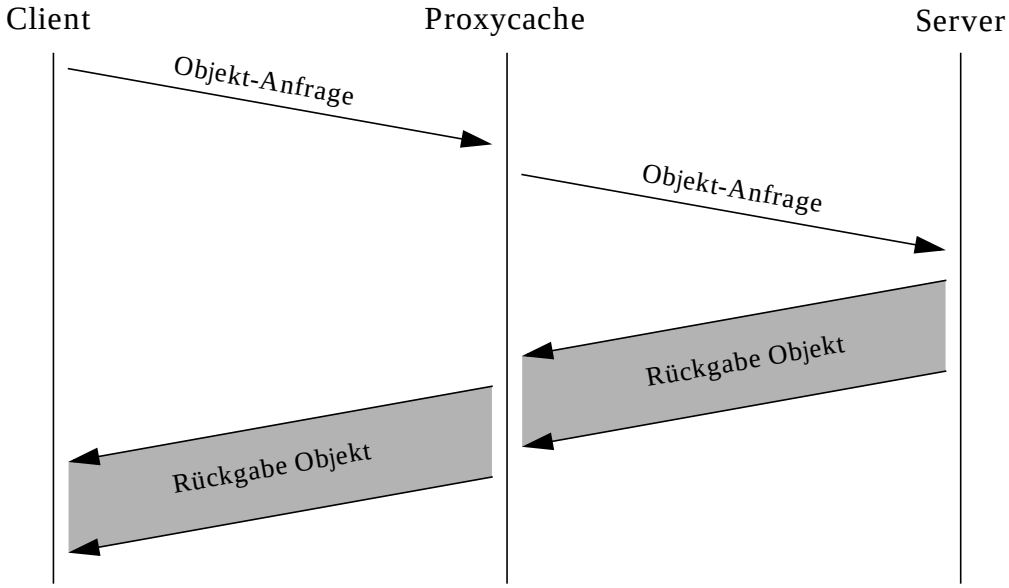

Abbildung 2.3: Client-Server-Kommunikation über einen Proxycache (MISS)

1. Hier sind hauptsächlich die Einsparung an benötigter Bandbreite bzgl. des firmenexternen Netzes (Extranet) sowie die Belastung der externen Geräte wie z.B. Server und Router gemeint.

Befindet sich jedoch das fragliche Objekt schon im Cache (HIT, siehe Abbildung 2.4), fällt der gesamte Verbindungsauf- und abbau zum Server, sowie die Objekt-Anfrage und Empfang des Objekts seitens des Proxycaches komplett weg: Das Objekt wird stattdessen direkt vom Cache gelesen und an den Client geschickt.

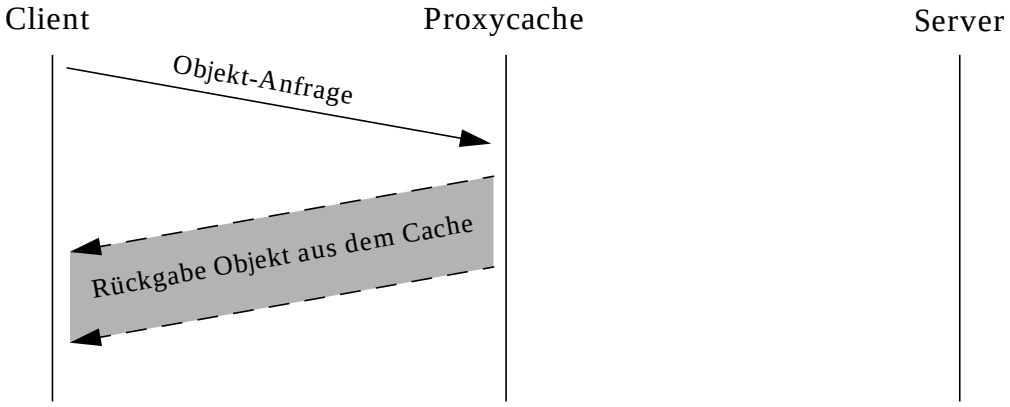

Abbildung 2.4: Client-Server-Kommunikation über einen Proxycache (HIT)

Somit wird durch den Einsatz eines Proxycaches im Falle eines HIT bereits Bandbreite bzgl. der nun nicht benötigten Verbindung zwischen Client und Server eingespart und die Verfügbarkeit der angeforderten Objekte erhöht, sowie der entsprechende Server entlastet.

2.2 Cache Hierarchien

2.2.1 Notwendigkeit eines Proxycache-Verbunds

Weitere Überlegungen sowie Forschungen im Rahmen des *Harvest research projects* [14] und letztendlich auch die Praxis haben gezeigt, daß es unter gewissen Voraussetzungen durchaus sinnvoll ist, mehr als nur einen Proxycache zu benutzen bzw. mehrere Proxycaches zu einem Verbund zusammenzuschließen (unter Zusammenschluß soll hier in erster Linie verstanden werden, daß jeder Proxycache auf den Inhalt der anderen Proxycaches in diesem Verbund zugreifen kann). Solche Voraussetzungen können z.B. sein:

- relative Ressourcenknappheit eines Proxycaches (RAM-Speicher, Festplattengröße, nutzbare Bandbreite bzgl. Netzzugang)
- schnellere Verbindung (höhere Bandbreite) zu den anderen Proxycaches im Vergleich zu den i.d.R. zu kontaktierenden Servern.

- kostengünstigere Verbindung zu den anderen Proxycaches als zu den i.d.R. zu kontaktierenden Servern (i.w. auch als Originalserver bezeichnet)
- höhere Ausfallsicherheit (Clients können bei Ausfall eines Caches einen anderen nutzen)
- Entlastung eines oder mehrerer Proxycaches
- einfacherer Lastausgleich (load balancing) über mehrere Leitungen (Links) möglich

2.2.2 Notwendigkeit einer Proxycache-Hierarchie

Sind relativ viele Proxycaches in einem Verbund zusammengeschlossen und entspricht dessen Topologie einem vermaschten Netz, kann es ohne dem Treffen notwendiger Vorkehrungen zu einer Endlosschleife bzgl. der Anfrage nach bestimmten Objekten (Request Loop) innerhalb des Proxycacheverbunds kommen (z.B. Proxycache 1 -> Proxycache 2 -> ... -> Proxycache N -> Proxycache 1). Moderne Software, wie z.B. Squid [15] erkennen zwar solche Request Loops, die Suche nach der Ursache des Problems und dessen Lösung bleibt jedoch den Administratoren überlassen, da es dafür derzeit keinen effizienten Algorithmus gibt, der dies automatisch bewerkstelligen könnte, ohne die evtl. daraus resultierende Verschlechterung der Effizienz eines oder mehrerer Proxycaches innerhalb eines Verbunds mit Sicherheit verhindern zu können. Um Request Loops vermeiden, als auch die Anfragen nach Objekten innerhalb eines Proxycache-Verbunds besser steuern zu können, muß die Funktion aller Proxycaches in diesem Verbund wohl definiert sein und die Anfrage eines Proxycaches nach einem bestimmten Objekt an einen oder mehrere andere Proxycaches nach festgelegten Regeln bzw. Reihenfolgen erfolgen. D.h. der Verbund muß die Form einer Hierarchie annehmen. In allen bisher bekannten Cache-Hierarchien[1] werden i.d.R. folgende Begriffe zur Beschreibung der Funktion eines Proxycaches innerhalb der Hierarchie verwendet[2]:

- *Parent*: stellt ein Proxycache eine Anfrage an seinen Parent, *muß* der Parent das angeforderte Objekt zurückliefern. D.h., wenn sich das angeforderte Objekt nicht im Cache des Parents befindet (*MISS*), muß dieser das Objekt entweder durch Anfrage an andere Proxycaches oder dem entsprechenden

1. siehe auch Web Caching Projects unter http://www.w3.org/Propagation/
2. Bisher wurden diese Begriffe in keinem RFC definiert. Sie beruhen weitestgehend auf der Proxycache- Software **Squid** [15], die jene Begriffe zur Konfiguration des Proxycaches nutzt. Auf die Übersetzung der Begriffe ins Deutsche wird bewußt verzichtet.

Server herunterladen. Er hat damit eine übergeordnete Stellung bzgl. des anfragenden Proxycaches (befindet sich eine Ebene höher).

- *Sibling*: stellt ein Proxycache eine Anfrage an seinen Sibling, liefert dieser das angeforderte Objekt nur zurück, wenn es sich bereits in seinem Cache befindet (*HIT*). D.h., bei einem MISS wird kein weiterer Proxycache bzw. Server von ihm befragt. Er ist in der Hierarchie dem anfragenden Proxycache gleichgestellt (befindet sich auf der gleichen Ebene).
- *Neighbor*: ist ein Parent oder Sibling.
- *Child*: Proxycache, der eine Anfrage an seinen Parent schickt und diesem somit unterstellt ist (befindet sich eine Ebene tiefer).
- *Root-Cache*: Parent, der in der Cache-Hierarchie auf der obersten Stufe steht, d.h. keine weiteren Parents besitzt.

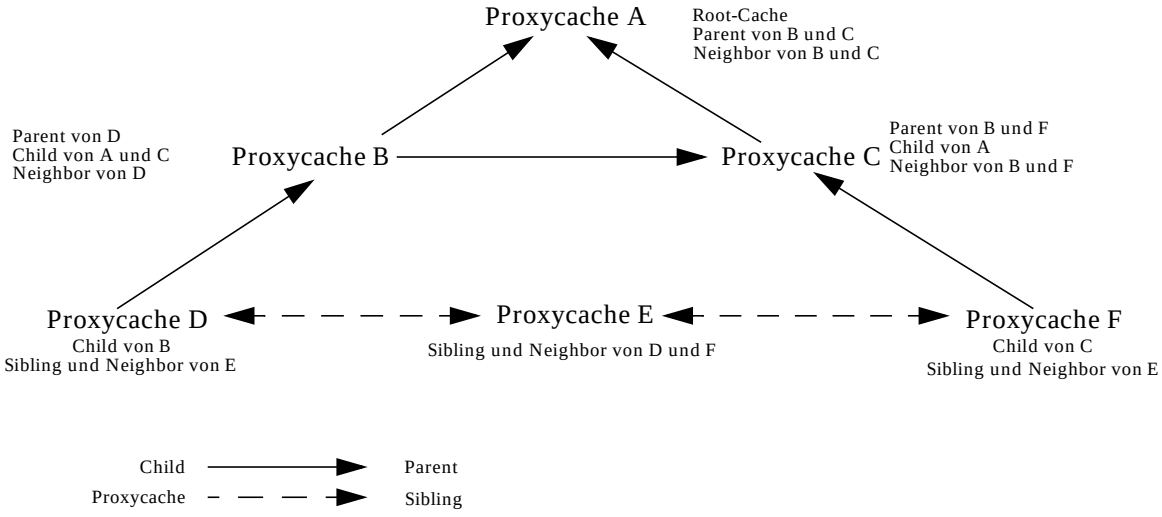

Abbildung 2.5: Elemente einer Proxycache-Hierarchie

Zusätzlich zu den oben genannten Begriffen sei hier der Begriff *Cache-Set* definiert, welcher die Menge aller WWW-Objekte beschreibt, die durch eine oder mehrere Regeln eines Proxycaches wohl definiert ist. So kann ein Cache-Set z.B. die Menge aller WWW-Objekte bezeichnen, deren URLs (siehe Anhang A.4) im host part auf „.edu“ enden.

Abbildung 2.5 veranschaulicht beispielhaft eine Hierarchie mit den entsprechenden Elementen. Es sei ausdrücklich darauf hingewiesen, daß jeder Proxycache mehrere Parents, Siblings und auch Children haben und er selber auch Sibling seiner Siblings sein kann. Ebenso muß beachtet werden, daß sich die oben genannten Begriffe oftmals nur auf einen bestimmten Cache-Set beziehen. D.h., es ist auch möglich, daß z.B. ein

Proxycache Root-Cache für alle Objekte, deren host part in der URL (siehe auf „.edu“ endet, jedoch Child für alle Objekte, deren host part in der URL nicht auf „.edu“ endet ist.

Um nun oben genannte Request-Loops zu vermeiden, muß in einer Cache-Hierarchie darauf geachtet werden, daß, sowie ein Proxycache mind. einen Parent besitzt, der Proxycache (MISS bei allen relevanten Neighbors in der Hierarchie) nicht Parent von sich selber sein darf.

Um Request-Loop verursachende Proxycaches innerhalb einer Hierarchie ausfindig zu machen, kann die im folgenden erarbeitete Vorgehensweise genutzt werden. Dabei ist es notwendig, die Hierarchie als gerichteten Graphen $G = (V,E,\varphi)$ mit folgenden Parametern zu betrachten:

- V: Menge der Neighbors (nur Parents und Children respektive)
- E: Menge der Verbindungen
- $(x,y) \in \varphi \Leftrightarrow$ x ist Child von y mit $x,y \in V$

Da Siblings aufgrund ihres Verhaltens keine Request Loops verursachen können, können sie aus V bzw. alle Verbindungen zu Siblings aus E ausgeschlossen werden. Weiterhin ist es sinnvoll festzulegen, daß ein Proxycache nicht sich selber als Neighbor definiert (Schlinge). Damit ist klar, daß der resultierende Graph unter Berücksichtigung der zuvor genannten Festlegung ein schlichter Graph sein muß. Also kann dieser Graph auch durch eine Adjazenmatrix A dargestellt werden, die nur die Elemente 0 (ist nicht Child von) oder 1 (ist Child von) und auf ihrer Hauptdiagonalen nur Nullen enthält. D.h., sie kann auf einem Rechner auch bitweise gespeichert werden (wichtig für sehr große Matrizen). Tabelle 2.1 zeigt eine solche Adjazenzmatrix (i.w. als Cache-Matrix C bezeichnet), die einen Ausschnitt aus der derzeitigen DFN[1]-Cache-Hierarchie darstellt.

Um nun Proxycaches zu finden, die evtl. Loop-Requests verursachen könnten, kann der „Depth-first search“ Algorithmus benutzt werden [11].

Während man die Hierarchie auf diese Weise traversiert, muß man sich nur den bisherigen Weg zu dem aktuellen Knoten merken. Trifft man im weiteren Verlauf auf einen

1. DFN ... Verein zur Förderung eines deutschen Forschungsnetzes e.V. (siehe http://www.dfn.de/dfn/dfn-portraet.html)

bereits überquerten Knoten, ist in dem Graphen eine gerichtete Kantenfolge vorhanden und somit ein auftretender Request Loop nicht auszuschließen. An dieser Stelle muß der benutzte Algorithmus gewährleisten, daß hier „umgekehrt“ wird.

φ	**leipzig.www-cache.dfn.de**	**berlin.www-cache.dfn.de**	**proxycache.cs.uni-magdeburg.de**	**WWW-cache.uni-magdeburg.de**	**proxy.med.uni-magdeburg.de**	**mkdir.boerde.de**	**proxycache1.cs.uni-magdeburg.de**
leipzig.www-cache.dfn.de	0	0	0	0	0	0	0
berlin.www-cache.dfn.de	0	0	0	0	0	0	0
proxycache.cs.uni-magdeburg.de	1	1	0	0	0	0	0
WWW-cache.uni-magdeburg.de	0	0	1	0	0	0	0
proxy.med.uni-magdeburg.de	0	0	1	1	0	0	0
mkdir.boerde.de	0	0	1	0	0	0	0
proxycache1.cs.uni-magdeburg.de	0	0	1	0	0	0	0

Tabelle 2.1: Auszug aus der derzeitigen DFN-Cache-Hierarchie

Der oben beschriebene Algorithmus funktioniert natürlich nur, wenn alle Proxycaches den oder die gleichen Cache-Sets benutzen. Ist dies nicht der Fall, was aufgrund vielfältiger und teilweise sehr unterschiedlichen Prämissen der einzelnen Proxycache-Betreiber die Regel sein dürfte, muß der Algorithmus abgewandelt werden. Idee dabei ist, alle Cache-Sets auf folgende Art und Weise in diskjunkte Cache-Sets zu überführen und dann die obige Prüfung für jeden einzelnen durchzuführen:

(1) Man verwende für jeden in der Hierarchie verwendeten Cache-Set eine eigene Cache-Matrix, initialisiere sie entsprechend und stelle sich alle Cache-Matrizen hintereinander angeordnet (Z-Achse) vor. Existieren m verschiedene Cache-Sets und n zu analysierende Proxycaches, hat diese neu entstandene Matrix die Dimension $n \times n \times m$.
Für diesen sowie alle weiteren Schritte muß gelten:

(a) i, j, k mit $1 \le i \le m$, $1 \le j \le m$, $1 \le k \le m$, $i \ne j$

(b) a, b mit $1 \le a \le n$, $1 \le b \le n$

(c) die zu Cache-Set$_i$ zugeordnete Cache-Matrix sei C_i

(d) die Anordnung der Relation der Proxycaches zueinander ist in jeder Cache-Matrix gleich, d.h. das Ergebnis der Relation „Proxycache$_a$ ist Child von Proxycache$_b$“ bzgl. Cache-Set$_i$ befindet sich in der Matrix an der Position (a, b, i), dann folgt daraus auch, daß das Ergebnis der Relation „Proxycache$_a$ ist Child von Proxycache$_b$“bzgl. Cache-Set$_j$ an der Position (a, b, j) zu finden ist.

(e) die Operation $C_i \oplus C_j = C_k$ sei als elementeweise ODER Verknüpfung der Matrizen C_i und C_j zu C_k für alle a, b lt. (1.b) entsprechend Tabelle 2.2 erklärt.

c_{abi}	c_{abj}	c_{abk}
0	0	0
0	1	1
1	0	1
1	1	1

Tabelle 2.2: ODER Verknüpfung von Cache-Set-Matrizen: Wertetabelle

(2) Es ist jeder vorhandene Cache-Set mit allen anderen Cache-Sets wie folgt zu vergleichen und falls notwendig die angeführten Operationen auszuführen:

(a) wenn Cache-Set$_i$ = Cache-Set$_j$, dann $C_i \oplus C_j = C_i$ und lösche C_j; ansonsten

(b) wenn Cache-Set$_i \cap$ Cache-Set$_j = \varnothing$, dann bleiben C_i und C_j unverändert erhalten; ansonsten

(c) wenn Cache-Set$_i \subset$ Cache-Set$_j$; dann $C_i \oplus C_j = C_j$ und

(3) gehe zu Schritt (2) bis bzgl. aller Cache-Sets mit $i \ne j$ gilt:

(a) Cache-Set$_i$ ist Teilmenge von Cache-Set$_j$ *oder*

(b) Cache-Set$_i$ ist disjunkt zu Cache-Set$_j$

(4) Man führe für alle Cache-Sets nacheinander folgende Operation solange durch, bis sich keine Cache-Matrix mehr ändert:
Besteht eine „ist Child von“ Relation zwischen zwei Proxycaches bzgl. eines Cache-Sets, prüfe man, ob diese Relation auch bzgl. aller anderen Cache-Sets gilt (ist Cache-Set$_i$ Teilmenge von Cache-Set$_j$, so ist Proxycache$_a$ auch Child von Proxycache$_b$ bzgl. Cache-Set$_j$, wenn Proxycache$_a$ Child von Proxycache$_b$ bzgl. Cache-Set$_i$ ist). Falls dies zutrifft, trage man eine 1 an der entsprechenden Position (a, b, j) ein.

(5) Man wende den „Depth-first search“ Algorithmus entsprechend an (durch die zusätzlichen Matrizen auf der Z-Achse kann jetzt ein Knoten natürlich mehr adjazente Knoten, als im obigen Beispiel mit nur einer Cache-Matrix besitzen).

Damit steht ein Algorithmus zur Verfügung, der auch in großen Hierarchien mit mehreren verschiedenen genutzten Cache-Sets potentielle Proxycaches für Request Loops ausfindig machen kann.

Da die verwendeten Cache-Sets sehr unterschiedlich sein können, muß in Betracht gezogen werden, daß die Schritte (1) bis (4) nur teilweise automatisiert werden können und die Interaktion des Nutzers benötigt, da derzeit kein geeigneter Algorithmus bekannt ist, der vorhandene Relationen zwischen unterschiedlichsten Cache-Sets bestimmen könnte. Deshalb ist dieser Algorithmus für relativ kleine, „übersichtliche“ Cache-Hierarchien sicher zu hoher Aufwand bzgl. der aufzuwendenden Zeit vom Nutzer zur Eingabe der Relationen für die einzelnen Cache-Sets. Wird die Hierarchie allerdings relativ groß und unübersichtlich und besitzt Proxycaches mit mehreren unterschiedlichen Cache-Sets, kann die Suche nach möglichen Quellen für Request Loops sicher nur noch unter Zuhilfenahme eines Algorithmus (z.B. des oben beschriebenen, im Rahmen dieser Diplomarbeit entwickelten) gelöst werden.

2.2.3 Cache-Protokolle

Um die Interaktion zwischen Proxycaches und damit z.B. den Aufbau eines Cache-Verbundes oder einer Cache-Hierarchie zu ermöglichen, sowie die Kommunikation in selbigen zu effektivieren, wurden verschiedene Cache-Protokolle entwickelt. Im folgenden soll ein kurzer Überblick über einige ausgewählte Protokolle gegeben werden, die erkennen lassen, daß auch hier weitere Forschungen notwendig sind, um den erzielten Nutzen aus der Verwendung von Proxycaches in vermaschten oder hierarchisch gegliederten Verbunden zu erhöhen.

2.2.3.1 Das Internet Cache Protocol (ICP)

Das derzeit wohl bekannteste Cache-Protokoll ist das Internet Cache Protocol (RFC 2186 [12]). Es wurde als zentraler Bestandteil eines hierarchischen Cachesystems im Harvest research project [14] (oftmals auch als “Mutter aller Proxycaches” bezeichnet) entwickelt, fand in dessen Weiterentwicklungen (NetCache = kommerzielle Version [16]; Squid = frei verfügbare Version, deren Weiterentwicklung weiterhin staatlich gefördert wird [15]) weitere Verbesserungen und hat seit September 1997 den Status eines RFC (Request For Comments). Wahrscheinlich ist es der relativ frühen Verfügbarkeit (Anfang 1996) sowie der weiten Verbreitung der Harvest-Cache-Deri-

vate zu verdanken, daß die meiste Proxycache-Software dieses Protokoll zumindest teilweise unterstützt.

Dieses Protokoll spezifiziert ein Nachrichtenformat für die Kommunikation zwischen Proxycaches. Es wird hauptsächlich dazu genutzt, um WWW-Objekte innerhalb eines Cache-Verbunds zu lokalisieren (ICP_OP_QUERY) bzw. die round trip time (RTT) zu einem Originalserver (ICP_OP_SECHO) oder WWW-Cache (ICP_OP_DECHO), der nicht ICP versteht, zu bestimmen. Es erlaubt weiterhin einem Cache dem anfragenden Cache zu signalisieren, daß:

- die Anfrage nicht verstanden wurde (ICP_OP_INVALID, ICP_OP_ERR)
- das angeforderte Objekt im Cache [nicht] vorhanden ist (ICP_OP_MISS, ICP_OP_HIT) und Heruntergeladen werden darf (im Falle eines MISS muß der gefragte Cache das Objekt von einem anderen Cache oder direkt vom Originalserver herunterladen)
- das Herunterladen des angeforderten Objektes nicht erlaubt ist (ICP_OP_DENIED)
- das Herunterladen des angeforderten Objektes derzeit nicht möglich ist (ICP_OP_NOFETCH)
- das angeforderte Objekt sich bereits im Datenteil der Antwort (query response) befindet (ICP_OP_HIT_OBJ). Dies ist nur erlaubt, wenn das Objekt in ein einziges Datenpaket paßt (meistens ist dies die vom Betriebssystem voreingestellte max. Größe für UDP-Pakete) und der anfragende Cache dies durch das Setzen eines entsprechenden Flags in der Anfrage-Nachricht erlaubt (ICP_FLAG_HIT_OBJ)
- die Antwort die von ihm zum Originalserver gemessene RTT enthält (bedarf das Setzen des Flags ICP_FLAG_SRC_RTT in der Anfrage).

Da dieses Protokoll relativ einfach ist, gestattet es oftmals eine effizientere Kommunikation und mehr Möglichkeiten zum Informationsaustausch zwischen Caches als z.B. mittels HTTP. Derzeit nutzen alle bekannten Implementierungen von ICP hauptsächlich aufgrund besserer Effizienz UDP, wobei darauf hingewiesen wird, daß die Spezifikation Implementierungen nicht auf UDP beschränkt. Beispiele für derzeitige Nutzungen von ICP in Applikationen können im RFC 2187 [13] studiert werden.

Leider hat dieses Protokoll auch Nachteile. So kann z.B. beim Versenden einer ICP_OP_SECHO oder ICP_OP_DECHO zwar die Erreichbarkeit der entsprechenden Maschine und die entsprechenden RTT ermittelt werden, doch bedeutet eine empfangene Antwort (Acknowledgement) nur, daß der ECHO-Server der Maschine funktio-

niert, nicht aber, das der Cache- bzw. WWW-Dienst läuft oder das anzufordernde Objekt tatsächlich schneller als von einem anderen Proxycache oder dem Originalserver ausgeliefert wird. Ein weiteres Problem ist die Nutzung des ICP_FLAG_HIT_OBJ - Flags. Da ICP die Standard-HTTP-Bearbeitung umgeht, d.h. einfach das angeforderte Objekt in die Antwort auf die Anfrage (query response) einbettet und sendet, wird z.B. auch keine Validierung bzgl. Alter (age) oder Autorisation (authorization) vorgenommen und es damit zur Auslieferung nicht mehr aktueller Objekte bzw. privater Objekte an unautorisierte Nutzer kommen kann. Dies ist der Hauptgrund, warum die Autoren des IPC Protokolls heute vom Gebrauch dieser Option abraten (siehe RFC 2168 [12] und RFC 2187 [13]). Ein weiterer Seiteneffekt ist, daß die max. Größe einer query response (abhängig vom verwendeten Betriebssystem, z.B. bei Solaris 2.5.1 16 KB, bei Solaris 2.6 8 KB) oftmals größer als die maximale Paketgröße für eine Übertragung (path Maximum Transmission Unit - MTU) ist, die beim Ethernet 1504 Bytes, bei ATM 9103 Bytes beträgt. Somit ist es oftmals notwendig, daß Antworten mit einem eingeschlossenen WWW-Objekt fragmentiert werden müssen, und damit Verzögerungen verursachen. Da derzeit ICP ausschließlich über UDP verwendet wird, besteht auch eine höhere Wahrscheinlichkeit, daß eines dieser Pakete verloren geht und das entsprechende Objekt von der Applikation noch einmal angefordert werden muß.

Als letzter Nachteil sei hier das Fehlen einer Autorisierungmethode für Client-Caches genannt. Applikationen ist es nur möglich, die Quell-IP-Adresse (source address) mit denen in deren Zugriffssteuerungsliste (access control list - ACL) zu vergleichen. Ist die IP-Adresse nicht darin enthalten, sollte die ICP-Anfrage ignoriert werden. Da aber eine Autorisierungsmöglichkeit in ICP fehlt, kann es immer noch durch sogenanntes IP-Spoofing (verfälschen der Absender-Adresse) überlistet werden und sieht den IP-Spoofer als ganz normalen („autorisierten“) Client an. Ausgehend davon sind weitere Angriffe möglich, die im folgenden nur kurz aufgezählt werden. Detailliertere Ausführungen können im RFC 2187 [13] nachgelesen werden:

- Einfügen von fehlerhaften ICP-Anfragen
- Einfügen von fehlerhaften ICP-Antworten
- Verhinderung, daß ein bestimmter Neighbor genutzt wird
- Anweisung, daß ein bestimmter Neighbor genutzt wird
- Cache-Vergiftung (cache poisoning) (Auswechseln / Veränderung der Daten bei ICP_OP_HIT_OBJ Antworten)

- Eavesdropping (Ausspionieren von gesendeten Nachrichten)
- Blockierung von ICP Nachrichten (Verhinderung der Nutzung eines o. mehrerer Neighbors)
- Verzögerung von ICP Nachrichten (schlechtere RTT führt u.U. zur Auswahl ungeeigneter Neighbors)
- Dienstblockierung (denial-of-service attack) durch kontinuierliches Senden von fehlerhaften ICP Nachrichten (Plattenplatz für Logdateien, socket receive queue, CPU Zeit)
- Veränderung von ICP Feldern (Opcode, Version, Nachrichtenlänge, Anfragenummer)

2.2.3.2 Das Web Cache Control Protocol (WCCP)

Das WCCP ist proprietärer Standardbestandteil der CISCO Internet Operation Software (Cisco IOS^{TM}), welches die Kommunikation zwischen Cisco Routers und Cisco Cache Engines ermöglicht. Cisco Cache Engines sind dabei die mit einem speziellen Betriebssystem inkl. Proxycache-Software und Festplatten ausgestattete Geräte, die die Funktion eines Proxycaches für einen festgelegten Cache-Set haben.

Grundprinzip ist, daß alle WWW-Anfragen (Datenpakete mit TCP destination port 80 = HTTP standard port) von einem CISCO Router mit laufendem WCCP nicht zur Zieladresse, sondern über ein lokales Subnetz zu einem Cache Engine umgeleitet werden (redirect). Dabei bestimmt der Router (home router), ob Cache Engines verfügbar sind und leitet gegebenenfalls Anfragen zu einem Cache Engine um. Besitzt ein Cache Engine nicht das angeforderte Objekt, fragt dieser alle anderen in der Installation befindlichen Caches (Cache Farm) danach. Besitzt keiner der Cache Engines das Objekt, wird dieses vom Originalserver heruntergeladen, im Cache Engine abgespeichert und an den anfragenden Client ausgeliefert.

Durch das WCCP wird der gesamte IP-Adressraum in 256 diskrete Gruppen unterteilt und dynamisch auf alle verfügbaren Cache Engines gleichmäßig aufgeteilt. Es stellt sicher, daß ein Cache Engine immer die Anfragen für den gleichen Originalserver im Internet beantwortet. Wird ein neuer Cache Engine einer Cache Farm hinzugefügt, informiert das WCCP den Cisco Router, daß ein neuer Cache Engine zur Verfügung steht. Daraufhin werden die 256 diskreten IP-Adress-Gruppen reallokiert, gleichmäßig auf die zur Verfügung stehenden Cache Engines neu aufgeteilt und Anfragen entsprechend umgeleitet. Gleichzeitig sendet der hinzugefügte Cache Engine einen Broadcast

aus, der alle anderen Cache Engines nach Daten fragt, die zu seinen Adress-Untergruppen gehören. Daraufhin senden alle Cache Engines die entsprechenden Objekte zu diesem neuen Cache, welcher dann seinen Dienst aufnimmt. Fällt dagegen ein Cache Engine aus, werden dessen Adress-Untergruppen auf alle noch verbleibenden Cache Engines wiederum gleichmäßig aufgeteilt. Ist der Cache Engine wieder einsatzbereit, bekommt dieser wieder seine ursprünglichen Adress-Untergruppen zugewiesen und lädt auf dieselbe Art und Weise wie beim Hinzufügen eines neuen Cache Engines alle Objekte für diese Adress-Untergruppen von den übrigen Cache Engines herunter. Sind alle Cache Engines ausgefallen, leitet der Cisco Router alle HTTP Anfragen (Port 80) nicht mehr um, sondern sendet sie auf traditionelle Weise an den Originalserver.

Jeder Cache Engine überwacht sowohl die Hitraten als auch das Verkehrsaufkommen für jede seiner Adress-Untergruppen und nutzt diese Statistiken, um dynamisch stark frequentierte Adress-Untergruppen von intensiv zu weniger genutzten Cache Engines zu reallokieren und damit den Verkehr gleichmäßig auf alle Cache Engines zu verteilen (load balancing) [17].

Der Cache Engine gewährleistet die Aktualität (Freshness) eines Objekts durch die im HTTP 1.1 festgelegten Regeln. Nutzt ein Server HTTP 1.0, so wird darüber hinaus die Option „Pragma: no cache“ honoriert bzw. durch die Multiplikation des Alters des Objektes mit einem festgelegten Faktor (default 0.1) dessen Freshness berechnet. Text-Dokumente gelten generell max. 24 Stunden als fresh. Andere „Multipurpose Internet Mail Extension“ (MIME) -Typen inkl. Bilder werden max. eine Woche als fresh betrachtet.

Das WCCP besitzt ebenso wie das ICP keine Authentifizierungsmöglichkeiten, so daß es ebenfalls bzgl. IP-Spoofing verwundbar ist. Da sie aber direkt an Cisco Router angeschlossen sind, kann der Cisco-Router konfiguriert werden, Pakete auf IP-Address-Spoofing zu kontrollieren und gegebenfalls zu verwerfen.

Aus den von Cisco bisher veröffentlichten Dokumenten bzgl. des WCCP [17] ist zu schließen, daß es mittels WCCP nicht möglich ist, explizit Cache Engines anderer Cisco Router zu nutzen. Cisco schlägt vor, an Routern auf dem Weg zum Originalserver weitere Cache Engines zu installieren und somit eine Art Hierarchie zu bilden. Desweiteren soll es möglich sein, alle Anfragen einer Cache Farm zu einem „traditionellen“

Proxy-Server weiterzuleiten (forwarding) [18]. Der Hersteller weist darauf hin, daß in der Zukunft auch ICP unterstützt werden soll.

Zusammenfassend kann somit festgestellt werden, daß das WCCP unter Ausnutzung der IOS Software eines Cisco Routers einen Lastenausgleich zwischen den daran angeschlossenen Cache Engines (Proxycaches) als auch die Verfügbarkeit von WWW-Objekten beim Ausfall eines oder mehrerer Cache-Engines gewährleistet. Nachteilig ist, daß damit keine Cache-Hierarchien etabliert werden können und die proprietäre Cisco IOS Software auf den entsprechenden CISCO-Routern nicht zum Lastenausgleich für andere Proxycaches benutzt werden kann. Ebenso können nicht die verwendeten Cache-Sets verändert werden, da diese automatisch durch das Protokoll für die angeschlossenen Cache Engines bestimmt werden und somit auch automatisch den Einsatz innerhalb von Proxycache-Hierarchien verhindern. Es ist nicht bekannt, ob die Reorganisation der Cache Engines (z.B. wenn ein neuer Cache Engine hinzukommt oder reaktiviert wird und alle anderen Cache Engines die Daten für dessen Cache-Set übertragen) zu Engpässen bzgl. der zur Verfügung stehenden Bandbreite führt.

2.2.3.3 Das Cache Array Routing Protocol (CARP)

Das CARP (derzeitige Version ist 1.0 und liegt als Internet-Draft vor) dient der Aufteilung des URL-Raumes (URL space) auf verschiedene lose gekoppelte Proxy-Server, deren Gesamtheit vom Autoren als Cache Array bezeichnet wird. Damit ist es sowohl Proxy-Servern als auch WWW-Browsern (in diesem Abschnitt i.w. dem RFC entsprechend als Agenten bezeichnet) möglich, URL-Anfrage an den dafür vorgesehenen Proxy-Server zu senden und hilft somit Redundanz bzgl. zwischengespeicherter Objekte innerhalb eines Cache Arrays zu mindern. D.h., dadurch verschmilzt der Cache-Bereich aller im Cache Array verfügbaren Proxies zu einem gemeinsamen, großen virtuellen Cache, was sich positiv auf die insgesamt zu erwartende Hitrate auswirken dürfte. Des weiteren erlaubt das Protokoll load balancing zwischen den Proxies eines Cache Arrays und damit eine gleichmäßige Auslastung aller Proxies in selbigem.

CARP basiert auf:

(1) einer bekannten Proxy-Array-Mitgliedertabelle (proxy array membership table, i.w. PAMT) lose gekoppelter Proxies
(2) einer Hash-Funktion zur Aufteilung des URL space auf diese Proxies

wobei die PAMT als reine ASCII-Text-Datei definiert wird, die über eine Array Configuration URL bezogen werden kann. CARP spezifiziert nicht, wie diese Tabelle erzeugt wird oder werden kann, sondern nur deren Format und wie sie durch Clients genutzt werden kann [19].

Sobald ein Agent die PAMT geladen hat, wird eine Hash-Funktion und ein Routing-Algorithmus dazu benutzt, URL Anfragen an das entsprechenden Mitglied des Proxy Cache Array zu stellen. Dabei wird der Name eines jeden Cache-Mitglieds gehasht (MemberProxy_Hash), mit dem Schlüssel der gehashten URL (URL_Hash) zu je einem kombinierten Schlüssel (Combined_Hash) verknüpft und dieser mit dem relativen Lastfaktor-Multiplikator (X_k) des entsprechenden Cache-Mitglieds multipliziert. Die so errechneten Ergebnisse werden als „score“s bezeichnet. Die Anfrage wird an das Cache-Mitglied geschickt, für das der höchste score errechnet wurde. Ist dieser nicht erreichbar, wird die Anfrage zu dem Cache-Mitglied mit dem zweitgrößten score usw. gesendet.

Da der URL_Hash, Combined_Hash und Score für jede Anfrage und Cache-Mitglied bestimmt werden muß, führt die Implementation dieses Protokolls zu einer relativ rechenintensiven Applikation bzw. Teil einer Applikation. Da sehr oft das bitweise Rotieren von Hashes verwendet wird, eignen sich bzgl. Performance Prozessoren wie Intel x86 besser als SPARC- oder Alpha-Prozessoren, da die Intel x86 Prozessoren Maschinenbefehle für solche Operationen besitzen und weniger CPU-Takte zur Bestimmung des Ergebnisses der bitweisen Rotation als andere Prozessoren benötigen.

Da das Protokoll keine Authentifizierungsmechanismen besitzt, ist auch CARP z.B. nicht vor IP-Spoofing geschützt, was z.B. zum Verfälschen der vom den Agenten regelmäßig angeforderten PAMT genutzt werden kann. Weitere Angriffe, analog den bei der Darstellung des ICP Protokolls genannten, sind ebenso denkbar.

Abschließend kann festgestellt werden, daß dieses Protokoll nicht die Kommunikation der Caches untereinander bzgl. ihrer Inhalte erlaubt, sondern nur versucht, einen Lastenausgleich durch Routing zwischen verschiedenen Proxycaches bzgl. verschiedener Parameter zu erreichen. Damit wäre der Einsatz dieses Protokolls in der Software von Routern bzw. von Proxycaches zur Vermeidung von redundant zwischengespeicherten Daten innerhalb eines Verbundes oder innerhalb einer bestimmten Stufe in ei-

ner Proxycache-Hierarchie wünschenswert, um damit den effektiv zur Verfügung stehenden Platz für zwischenzuspeichernde Objekte vergrößern zu können.

2.2.3.4 Das Hyper Text Caching Protocol (HTCP)

Das Hyper Text Caching Protocol [20] liegt seit März 1998 als Internet-Draft vor. Es wurde entwickelt, um die Limitierungen von ICP zu umgehen. Die gravierendsten Unterschiede zu ICP sind, daß mittels HTCP Request- und Response-Header von HTTP-Anfragen und HTTP-Antworten ausgetauscht werden können, somit die Gültigkeit (Freshness) eines Objekts bestimmt und damit „Hits“ auf veraltete Objekte (stale objects) einschränken kann (zur Erinnerung: dies ist bei ICP Nachrichten nicht möglich, da diese **keine** Header enthalten) und eine Möglichkeit zur Authentifizierung der HTCP-Nachrichten vorhanden ist. Darüber hinaus erlaubt das Protokoll das Monitoring von Caches (d.h, Notifikation des Monitors, wenn Objekte im Cache hinzugefügt, aufgefrischt, ersetzt oder gelöscht wurden und den Grund dafür) sowie Cache-Objekte eines entfernten Caches zu ersetzen oder sogar zu löschen.

Mittels HTCP ist es **nicht** möglich, ähnlich wie bei ICP, komplette Objekte in HTCP-Nachrichten zu übertragen. Damit kann geschlossen werden, daß auf einem Neighbor, der via HTCP mit dem anfragenden Cache kommuniziert, bedeutend mehr Ressourcen benötigt werden als bei ICP: im Falle eines HIT müssen mit hoher Wahrscheinlichkeit zumindest die Header von dem Plattenspeicher eingelesen (was relativ zeitaufwendig sein dürfte) oder es müssen generell alle Header zwischengespeicherter Objekte im RAM gehalten werden, was bedeutend höhere Ansprüche bzgl. RAM-Ausstattung des Neighbors impliziert. Des weiteren ist in Betracht zu ziehen, daß HTCP-Nachrichten mindestens 20 Bytes länger als ICP-Nachrichten sind und somit etwas mehr Bandbreite in Anspruch nehmen.

Vom Prinzip her ist HTCP sicher ein Fortschritt gegenüber ICP, allerdings muß dafür eine geringere Performance gegenüber ICP in Kauf genommen werden [21].

2.2.3.5 Cache Digests

Die Verwendung von Cache Digests [22] ist ein neuartiges Verfahren für die Kooperation zwischen Proxycaches (auch als kooperatives Web-Caching bezeichnet), daß zum Zeitpunkt der Erstellung dieser Arbeit entwickelt, im Einsatz mit Squid getestet und evaluiert wird. Mit diesem Verfahren wird versucht, Nachteile von Protokollen wie

z.B. ICP und HTCP (z.B. unnötige Anfragen bzgl. nicht zwischengespeicherter Objekte, Verzögerungen bei Cache MISS) zu kompensieren.

Das Prinzip ist, daß jeder Cache eine Übersicht (Digest) über alle zwischengespeicherten Objekte besitzt und diese auf Anfrage seines Neighbors zu selbigem überträgt. Wird beim Neighbor nun ein Objekt angefordert, sucht dieser in allen Digests seiner Neighbors, ob dieses Objekt enthalten ist. Ist dies der Fall, wird das Objekt direkt vom entsprechenden Cache angefordert. Wird dagegen das entsprechende Objekt in keinem Digest gefunden, muß der Cache wie üblich dieses Objekt von einen Parent oder direkt vom Originalserver anfordern.

Im Prinzip sicher keine schlechte Idee, da jetzt z.B. der Einsatz von ICP oder HTCP bzgl. der Kommunikation über das Vorhandensein einzelner Objekte überflüssig wird, da der Cache diese Information bereits aus dem lokal gehaltenen Digest ermitteln kann. Damit kann einerseits Bandbreite gespart und andererseits die Antwortzeit eines Caches im Falle eines MISS verringert werden, da nicht wie im Falle von ICP oder HTCP, auf die negative Bestätigung der Nachbar-Caches gewartet werden muß, sondern das angeforderte Objekt sofort bei einem Parent bzw. dem Originalserver angefordert werden kann.

Allerdings hat dieser Ansatz auch seine Nachteile und Probleme wie z.B. die Größe des Digest sowie dessen Aktualität und Genauigkeit. Nimmt man ausgehend von den Erfahrungswerten der Entwickler an, daß die durchschnittliche Größe eines Objekts 13 KByte ist und nutzt beispielsweise eine Cache-Kapazität von 4 GB, können folglich ca. 315 K Objekte im Cache gespeichert werden. Beträgt die durchschnittliche Länge eines URL 50 Byte (wiederum Erfahrungswerte), so würde für ein Cache-Digest aller Objekte bei max. Auslastung der Cache-Kapazität ca. 15.75 MByte benötigt werden. Damit würde sich im Vergleich zu ICP oder HTCP ein weitaus höherer RAM-Bedarf auf dem Cache ergeben (Digest sollten aus Gründen der Effizienz ständig im RAM gehalten werden) als auch in Abhängigkeit der Größe, Anzahl und Häufigkeit von zu übermittelnden Digests, eine bedeutend größerer Datentransfer. Dies zu umgehen, hat man verschiedene Untersuchungen angestellt mit dem Ergebnis, MD5 [23] in Verbindung mit einem Bloom-Filter [24] für Cache Digest zu verwenden. Damit wird eine hohe Komprimierung des Digest erreicht (auf ca. 250 KB je 4 GB Cache Area), im gleichen Zug aber Fehler (d.h. falsche Ergebnisse) bei der Suche nach Objekten im Cache Digest

in Kauf genommen, da durch das verwendete Verfahren keine eindeutigen Schlüssel generiert werden.

Untersuchungen im Rahmen des Squid-Projektes (siehe [22]) haben beispielsweise ergeben, daß ein 20 GB Cache, der im Durchschnitt zwei ICP-Anfragen je Sekunde von jedem seiner Neighbors bekommt, vom Umstieg auf Cache Digests profitieren würde, wenn die Update-Periode des Digest auf den Neighbors größer oder gleich 1.15 Stunden ist. Ebenso wurde in diesen Tests festgestellt, daß mit dem derzeit verwendeten Verfahren 19% falsche Vorhersagen über den Cache-Inhalt von Neighbors gemacht wurden (18% davon bzgl. HIT, auch als „false hit“ bezeichnet). Erwähnswert ist, daß sich, zumindest bei den Tests, das Update der Cache Digests auf den Neighbors mit einer Periode von 1 Stunde nicht sonderlich stark auf die Rate der „false misses“ ausgewirkt hat (ca. 1%).

Als ein weiteres Problem stellt sich die Übertragung der Cache Digests dar. Hat man unter Verwendung von ICP bzw. HTCP relativ kontinuierlich geringen Verkehr, resultiert die Übertragung von Cache Digests dagegen in stark burstartigen Verkehr, was zu relativ hohen Verzögerungen bei der Bedienung von Client-Anfragen führen kann. Um eine Reduzierung des gesamten Verkehrs bzgl. Cache Digests zu erreichen, sollte untersucht werden, ob es möglich und wenn ja auch effektiver ist, nur die innerhalb einer Periode aufgetretenen Änderungen innerhalb eines Digests an den anfragenden Proxycaches zu übertragen.

Als letztes sei angemerkt, daß die Übertragung von Cache Digests besonders anfällig gegen Angriffe ist, da diese i.d.R. nur einmal pro Stunde übertragen werden und somit eine große Anzahl von Informationen mit einmal verfälscht werden kann. Deshalb sollten Applikation, die Cache Digests verwenden, Mechanismen zur Authentifizierung selbiger nutzen.

2.3 Squid

2.3.1 Was ist Squid

> *Squid ist ein sehr leistungsstarker Proxycache-Server für Web-Clients, welcher HTTP-, FTP- und Gopher-Daten-Objekte unterstützt. Im Gegensatz zu traditioneller Cache-Software bearbeitet Squid alle Anfragen in einem einzigen, nicht-blokkierenden, E/A-basierten Prozeß.*
>
> *Squid hält Metadaten und speziell Objekte im RAM, die in einer bestimmten Zeit*

sehr oft abgefragt werden (hot objects), speichert Ergebnisse von DNS-Anfragen (DNS lookups) zwischen, unterstützt nicht blockierende DNS lookups und implementiert negatives caching fehlgeschlagener Anfragen.

Squid unterstützt das Secure Socket Layer Protokoll [25], umfangreiche Zugriffssteuerungen (access control lists - ACL) und die komplette Aufzeichnung von Anfragen (request logging). Durch den Einsatz des leichtgewichtigen Internet Cache Protocol (ICP) können Squid Caches in Hierarchien oder vermaschten Netzen zur zusätzlichen Einsparung von Bandbreite angeordnet werden.

Squid besteht aus einem Haupt-Server-Programm squid, einem Domain Name System lookup Programm dnsserver, einem Programm zum Herunterladen von FTP-Daten ftpget, und einigen Client- und Management-Tools. Wenn squid gestartet wird, startet dieser eine konfigurierbare Anzahl von dnsserver Prozessen, die jeweils einen blockierenden Domain Name System (DNS) lookup ausführen können. Dies reduziert die Zeit, die der Cache auf ein DNS lookup warten muß. [Desweiteren startet squid eine konfigurierbare Anzahl von Redirectors, die eine entsprechende Anfrage an den Cache durch einen anderen URL ersetzen können, bevor der Cache die Anfrage bearbeitet. (d.A.)]

Squid ist eine Weiterentwicklung des ARPA-geförderten Harvest project[1].

Squid war eine der ersten, kostenlos und im Quellcode verfügbaren Proxycache-Software. Aufgrund dessen sowie seiner mächtigen Features und Konfigurationsmöglichkeiten, als auch der Möglichkeit, eigene Anpassungen am Quellcode vornehmen zu können, hat sich Squid zu der weltweit wohl am meisten eingesetzten Proxycache-Software-Lösung auf allen gängigen UNIX-Betriebsystemen (z.B. Solaris, AIX, HP-UX, FreeBSD, Linux) etabliert.

Im folgenden sind die Auswertungen der Logfiles von drei WWW-Servern der Fäkultät für Informatik der Otto-von-Guericke-Universität bzgl. Zugriffe via Proxies dargestellt. Sie scheinen zumindest für diese drei WWW-Server zu bestätigen, daß Squid eine der am meisten genutzte Proxycache-Software ist. Bei der Interpretation der Daten ist allerdings zu beachten, daß:

(1) erst ab Oktober 1997 auch die *Via* header fields (siehe RCF 2068 [9]) der Anfragen ausgewertet wurden. Vor der Standardisierung von HTTP in der Version 1.1 gab es keine Vorschrift, ob und wie Proxies dokumentieren sollten, daß sie in der entsprechenden Anfrage involviert sind. So hat sich stillschweigend durchgesetzt, dies in dem agent header field (siehe RCF 2068 [9]) einzutragen. Diese Felder wurden in den entsprechenden Logfiles vor Oktober 1997 ausgewertet. Da HTTP / 1.1 erst im Januar 1997 standardisiert

1. gesamter Abschnitt frei übersetzt aus „Squid Frequently Asked Questions“ [26]

wurde, wird angenommen, daß bis Januar 1997 keine oder nur sehr wenige Proxies (Squid ab Version 1.1beta21 = 22. November 1996) das Via header field benutzten. D.h., in den Statistiken ist der Zeitraum von Januar 1997 - September 1997 einschließlich mit Vorsicht zu genießen.

(2) es durchaus auch Proxies geben kann, die sich nicht an RFC 2068 (HTTP / 1.1 Punkt 14.44, S. 137) halten und keinen Via header field der Anfrage hinzufügen

(3) falls ein Via header field mehrere Einträge enthält, wurden alle diese Einträge ausgewertet und auf den entsprechenden Statistik-Seiten die Anzahl der Via header mit mehr als einem Feld separat ausgewiesen.

(4) Proxies Via header fields von Proxies innerhalb ihrer Organisation u.U. zu einem Eintrag zusammenfassen können

(5) durch den Einsatz der entsprechenden Software (Harvest / Squid) an der Universität Magdeburg selbst auch automatisch deren Häufigkeit bei Anfragen beeinflußt.

Prozentsatz	**WWW-Server**
3 - 7%	http / / ivs.cs.uni-magdeburg.de /
20 - 26%	http: / / www.cs.uni-magdeburg.de /
1 - 3%	http: / / java.cs.uni-magdeburg.de /

Tabelle 2.3: Prozentsätze von Anfragen universitätsinterner Proxycaches

So wurden im Zeitraum von 97 / 10 - 98 / 07 zwischen 1 - 26 % aller Anfragen von Proxies an die in Tabelle 2.3 aufgeführten WWW-Server durch Proxycaches der Universität Magdeburg gestellt.

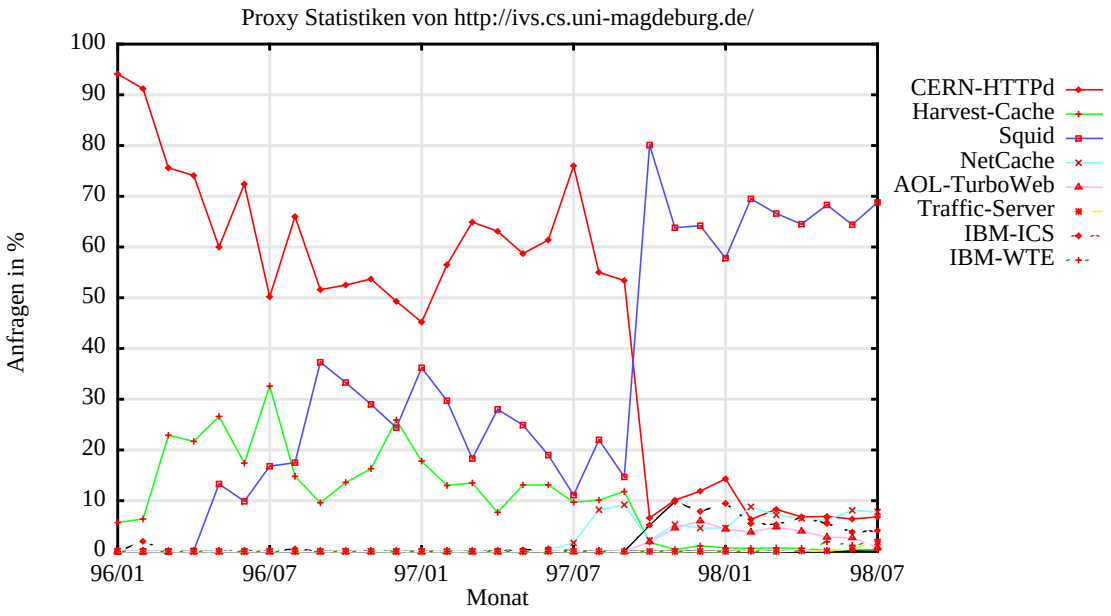

Abbildung 2.6: Verteilung von Proxy-Anfragen an http: / / ivs.cs.uni-magdeburg.de /

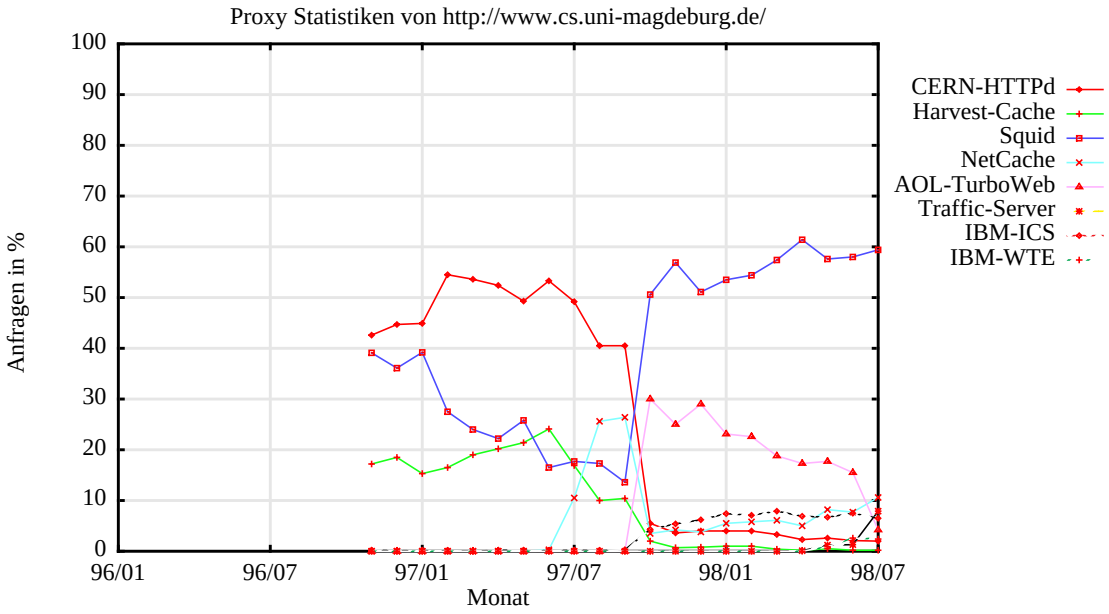

Abbildung 2.7: Verteilung von Proxy-Anfragen an http://www.cs.uni-magdeburg.de/

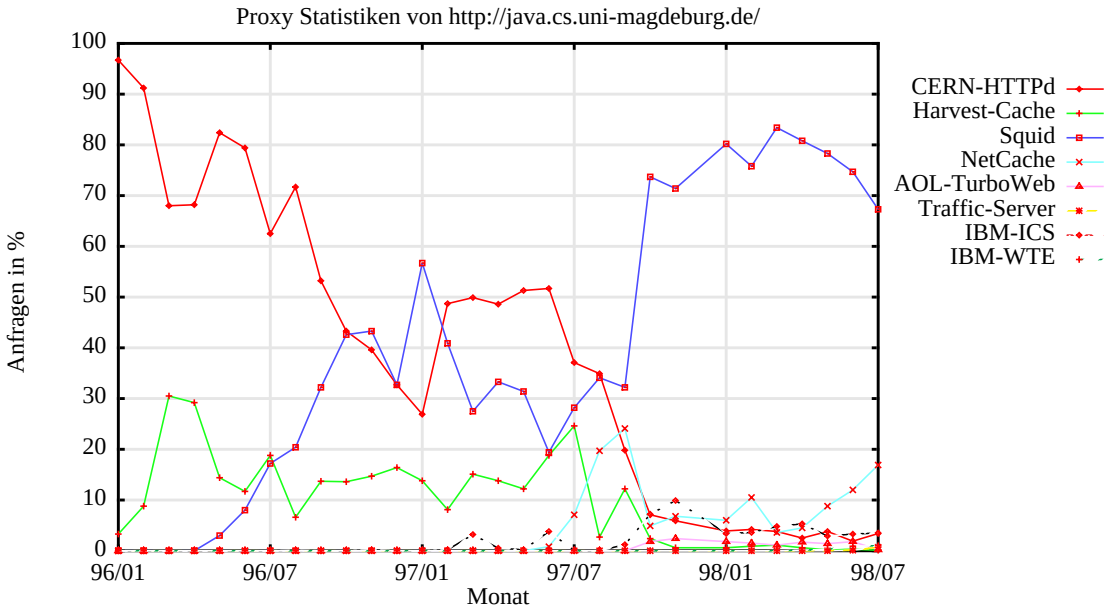

Abbildung 2.8: Verteilung von Proxy-Anfragen an http://java.cs.uni-magdeburg.de/

Generell ist auf Abbildung 2.6 bis Abbildung 2.8 zu erkennen, daß der Anteil der Anfragen via CERN-HTTPd-Proxies von fast 100% auf weniger als 5% Mitte 1998 abgesunken ist und somit geschlußfolgert werden kann, daß dieser in der Praxis nur noch sehr wenig Bedeutung bzgl. Web-Caching hat. Im Gegensatz dazu steht die Kurve von

Squid. Seit dessen freier Verfügbarkeit in der Version 1.0 im Frühjahr 1996 steigt dessen Anteil bis zu Herbst 1997 auf über 50% und stabilisiert sich oberhalb dieser Marke. Alle anderen Proxies kommen seit diesem Zeitpunkt i.d.R. nur noch sehr selten auf einen Anteil von über 10% aller Proxy-Anfragen an die genannten WWW-Server. Daraus wird geschlossen, daß Squid eine weitaus höhere Akzeptanz als andere Proxy bzw. Proxycache-Software-Lösungen bei vielen ISP gefunden hat und Untersuchungen zur Verbesserung dieser Software sich auch in naher Zukunft für sehr viele Betreiber von Squid-Proxycaches von Nutzen sein dürfte.

Der relativ hohe Anteil von AOL-TurboWeb bei www.cs.uni-magdeburg.de (siehe Abbildung 2.7) soll allerdings nicht unkommentiert bleiben: Ein Grund dafür ist, daß sich auf diesem Server die home pages des Deutschen XENA Fanclubs (siehe http://www.cs.uni-magdeburg.de/~rschulz/xena/dxf/) befinden, die mit Abstand die am meisten abgefragtesten Seiten auf diesem Server sind (siehe http://www.cs.uni-magdeburg.de/stats/WWW/), und die meisten Zugriffe auf diese Seiten via AOL Proxy erfolgen. Desweiteren befinden sich auf diesem Server die privaten home pages der an der Fakultät für Informatik immatrikulierten Studenten sowie allgemeine Informationen bzgl. der Fakultät. Der Server java.cs.uni-magdeburg.de enthält hauptsächlich Dokumentationen zum Thema Java und universitäre Software zum Download. Ivs.cs.uni-magdeburg.de ist der WWW-Server des Instituts für Verteilte Systeme und enthält hauptsächlich Seiten zu aktuellen Forschungsprojekten des Instituts sowie ergänzendes Material zu Vorlesungen und Seminaren.

2.3.2 Squid an der Universität Magdeburg

Um die im folgenden Kapitel angeführten Erfahrungen, Untersuchungen, Analysen und Ergebnisse, sowie die Grundlagen und Quellen der benutzten Daten besser einordnen zu können, widmet sich dieser Abschnitt der kurzen Darstellung der Entwicklung der Proxycache-Hierarchie an der Otto-von-Guericke-Universität Magdeburg.

Im April 1996 ein WWW-Cache-Projekt an der Fakultät für Informatik in Angriff genommen. Es wurden dazu die Harvest Cache-Software Version 1.4 sowie eine HP/UX 9000/715 (proxycache2.cs) der Fakultät für Informatik (FIN) und eine SPARCstation 20 des damaligen Institutes für Rechnernetze und Betriebssysteme (IRB) mit jeweils 64 MB RAM und jeweils 500 MB Cache Area (proxycache1.cs) eingesetzt. Die HP-Workstation wurde von allen im HP-Pool befindlichen Workstations als Proxycache benutzt,

sie selbst war proxy-only neighbor der Sun-Workstation, welche wiederum von allen im Sun-Pool sowie im IRB befindlichen Workstations als Proxycache benutzt wurde. Sie selber unterhielt bidirektionale Nachbarbeziehungen zu je einem Proxycaches der Freien Universität (FU) Berlin, der Technischen Universität (TU) Chemnitz, dem Deutschen Klimarechenzentrum (DKRZ) Hamburg als auch den Universitäten Leipzig, Bochum und Mainz.

Im September 1996 erfolgte der Umstieg auf Squid Version 1.0.12, da in diesem Monat die Beendigung des Harvest-Projektes bekannt gegeben wurde und Squid als dessen direkter Nachfolger feststand. Um den steigenden Anforderungen an die Ressourcen von Proxycaches gerecht zu werden, ermöglichte die Fakultät für Mathematik (FMA) die Nutzung einer weiteren HP-UX-Workstation mit 64 MB und 1 GB Cache Area als Proxycache (proxycache3.math). Desweiteren wurde eine Hamstation 19[1] mit 64 MB und 6 GB Cache Area der FIN im Rahmen des ATM-Testbed-Projektes als dedizierter Server für Proxycaches in Betrieb genommen (proxycache.cs). Inzwischen konnten bereits Erfahrungen bzgl. des Einsatzes von Proxycaches in vermaschten Netzen von uns als auch anderen Betreibern gesammelt werden, so daß entschieden wurde, zwischen allen derzeitigen Proxycaches der Universität Magdeburg einer Hierarchie (i.w. als Magdeburger Cache-Hierarchie, kurz MCH bezeichnet) zu etablieren (siehe Abbildung 2.9), proxycache.cs als Parent aller anderen Proxycaches der Universität Magdeburg zu nutzen und diesen in die im Aufbau befindliche DFN-Cache-Hierarchie [27] (siehe Abbildung 2.10) als Child zu integrieren.

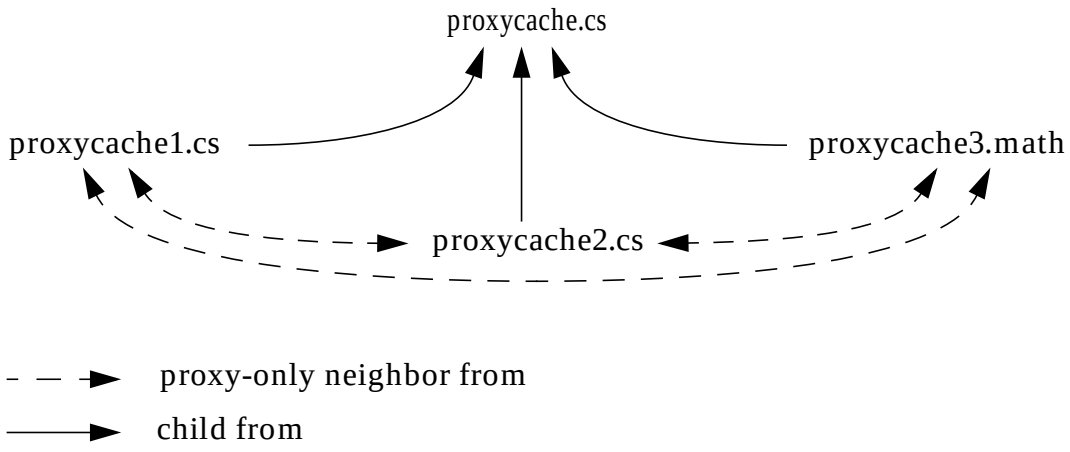

Abbildung 2.9: Proxycache-Hierarchie an der Universität Magdeburg im September 1996

1. SPARCstation 20 clone

Ebenso wurde alle bidirektionalen Beziehungen zu allen Nicht-DFN-Caches exklusive TU Chemnitz eingestellt. Die FU Berlin, die FSU Jena und das DKRZ Hamburg nutzen proxycache.cs weiterhin als Neighbor.

Im November 1996 installierte auch die Medizinische Fakultät der Universität Magdeburg einen Proxycache (proxycache4.med), und wurde analog den proxycaches 1-3 der Universität Magdeburg in die MCH integriert.

Im April 1997 setzte auch das Universitätsrechenzentrum (URZ) einen Proxycache auf (128 MB RAM, 1GB Cache Area), der Anfang März 1997 in die MCH analog den proxycaches 1-4 eingebunden wurde (proxycache0.urz).

Im August 1997 erhielt der proxycache.cs (Parent der MCH) eine weitere Festplatte sowie 256 MB RAM, so daß das zur Verfügung stehende Cache Area auf 12 GB ausgebaut und 384 MB statt 128 MB RAM genutzt werden konnten. Ebenso wurde der proxycache des URZ auf 384 MB RAM und 8 GB Cache Area aufgerüstet. Damit waren ausreichend Ressourcen vorhanden, um proxycache2.cs und proxycache3.cs von ihrer Funktion als Proxycache für die Universität Magdeburg zu entbinden und so eine Reduzierung der sogenannten Points of Failures sowie eine bessere Qualität zu erreichen, da diese Maschinen i.A. auch durch andere Dienste (WWW, File, NIS Server) teilweise sehr stark belastet wurden. Proxycache1.cs wird ab jetzt nur noch zum Test neuer Proxycache-Software und relevanter Tools genutzt, verbleibt aber wie bisher in der MCH.

Da die bisheriger Verteilung der cache_host_domains[1] innerhalb der DFN-Cache-Hierarchie (siehe Abbildung 2.10) zu bedeutenden Problemen bzgl. Last bei den Caches für die Top Level Domain (TLD) .com, d.h. Köln und Frankfurt geführt hat, wurden am 16. Oktober 1997 die cache_host_domains von deren Administratoren so aufgeteilt, daß 50% aller DFN-Caches die TLD .com und alle anderen alle TLDs zwischenspeichern, die nicht auf .com enden. Der Parent der MCH wurde so umkonfiguriert, daß ab nun die DFN-Caches Leipzig und Nürnberg Parents für die TLD .com,

1. Mittels dem TAG cache_host_domain kann in der Squid-Konfigurationsdatei für einen Neighbor ein Cache-Set festgelegt werden, wobei der als Parameter angegebene Domainname (z.B. uni-magdeburg.de) beim Parsen des host parts eines URLs benutzt wird, um festzustellen, ob ein Objekt zu dessen Cache-Set gehört, oder nicht. Alle Cache-Sets der DFN-Cache-Hierarchie basieren auf der Nutzung der Top-Level-Domain im host part von URLs als Kriterium für eine Cache-Set-Zugehörigkeit.

Berlin und Hamburg Parents für alle anderen TLDs sind. Aufgrund des schlechten Antwortverhaltens dieser Caches bei MISSes, werden diese ab November 1997 nur noch als Neighbors genutzt.

Seit Februar 1997 wird proxycache.cs als Parent des Börde e.V. (proxy.boerde.de) genutzt und proxycache0.urz.uni-magdeburg.de vom URZ in www-cache.uni-magdeburg.de umbenannt.

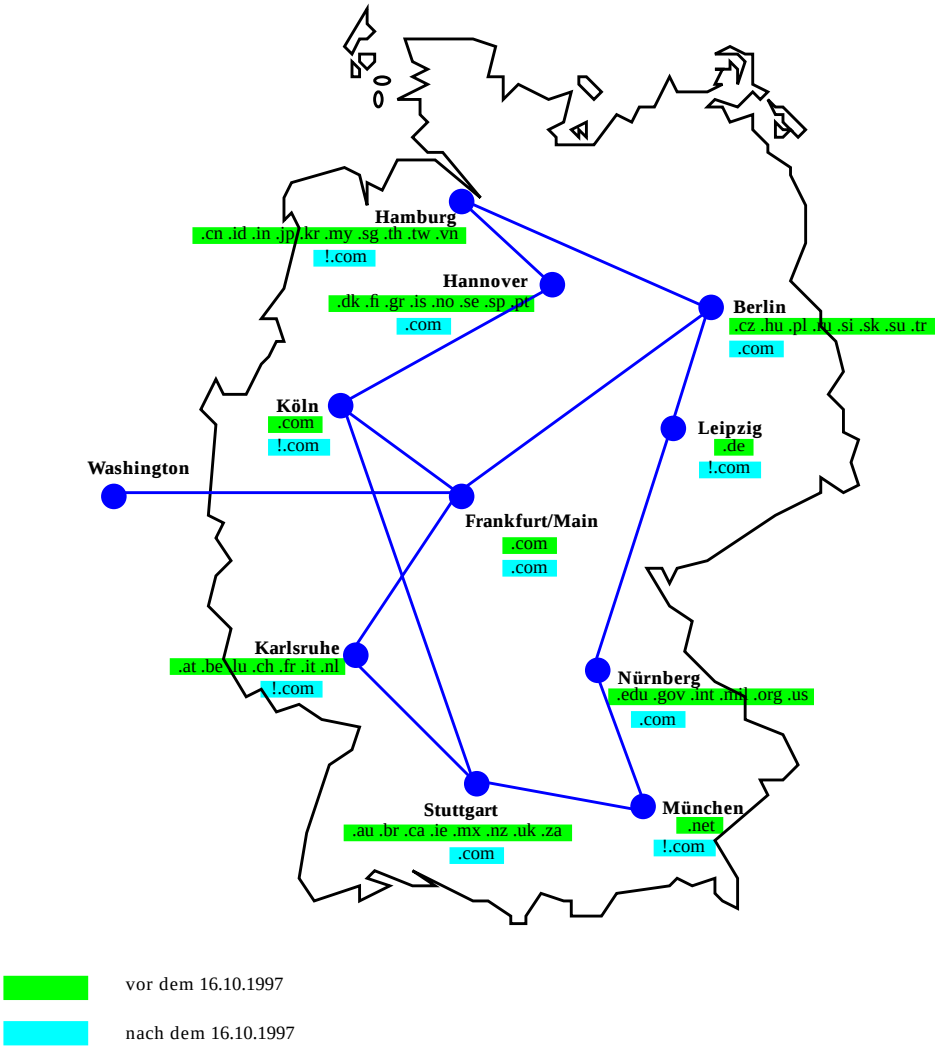

Abbildung 2.10: cache_host_domain - Aufteilung in der DFN-Cache-Hierarchie

Im Mai 1998 wird der Hauptspeicher des Parent der MCH um weitere 128 MB auf 512 MB RAM und im Juni 1998 das Cache Area auf 24 GB aufgerüstet. Damit ist die maximale Ausbaufähigkeit bzgl. Hauptspeicher erreicht und, wie nachfolgende Ana-

lysen und Berechnungen zeigen, aufgrunddessen der weitere Ausbau des Cache Area nicht mehr sinnvoll. Zur Unterdrückung von Werbung auf WWW-Seiten wird ab nun Jesred, ein Redirector für Squid [30], genutzt. Des weiteren wurde der Proxycache des Werner-von-Siemens Gymnasiums (siemens.sn.uni-magdeburg.de) in die MCH integriert. Der Proxycache des URZ nutzt ab nun den proxycache.cs nur noch als Sibling im proxy-only[1] Modus.

Damit gestaltet sich die derzeitige (Januar 1999) Konfiguration der MCH sowie seiner Beziehungen zu anderen Proxycaches wie in Tabelle 2.4 dargestellt.

... ist N von ... **mit N =** **-1 ...???** **0...nichts** **1...Child** **2...Neighbor** **3...Parent**	**proxycache.cs.uni-magdeburg.de**	**www-cache.uni-magdeburg.de**	**proxycache4.med.uni-magdeburg.de**	**siemens.sn.uni-magdeburg.de**	**proxy.boerde.de**	**proxycache1.cs.uni-magdeburg.de**	**leipzig.www-cache.dfn.de**	**berlin.www-cache.dfn.de**	**hamburg.www-cache.dfn.de**	**nuernberg.www-cache.dfn.de**	**www-cache.uni-jena.de**	**proxy.zrz.tu-berlin.de**
proxycache.cs.uni-magdeburg.de	0	2	3	3	3	3	1	1	1	1	2	2
www-cache.uni-magdeburg.de	0	0	2	0	0	2	0	0	0	0	0	0
proxycache4.med.uni-magdeburg.de	1	0	0	0	0	0	0	0	0	0	0	0
siemens.sn.uni-magdeburg.de	1	0	0	0	0	0	0	0	0	0	0	0
proxy.boerde.de	1	0	0	0	0	0	0	0	0	0	0	0
proxycache1.cs.uni-magdeburg.de	1	0	0	0	0	0	0	0	0	0	0	0
leipzig.www-cache.dfn.de	3	0	0	0	0	0	0	0	0	0	0	0
berlin.www-cache.dfn.de	3	0	0	0	0	0	-1	-1	-1	-1	-1	-1
hamburg.www-cache.dfn.de	3	0	0	0	0	0	-1	-1	-1	-1	-1	-1
nuernberg.www-cache.dfn.de	3	0	0	0	0	0	-1	-1	-1	-1	-1	-1
www-cache.uni-jena.de	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1
proxy.zrz.tu-berlin.de	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1

Tabelle 2.4: Beziehungen von und zu Proxycaches in der MCH

1. proxy-only bewirkt, daß alle Objekte, die vom entsprechenden Neighbor geholt, nicht zwischengespeichert werden.

3 Performance Tuning und Konsistenz von Cache-Objekten

In diesem Kapitel werden Vorschläge zur Konfiguration für Squid erarbeitet, die einen zuverlässigen Betrieb eines Proxycaches notwendig sind und dessen Leistungen verbessern helfen. Da hierbei auch die Konsistenz von zwischengespeicherten Objekten eine große Rolle spielt, wird auf die von Squid benutzten Regeln für das Caching von WWW-Objekten und den Einfluß von WWW-Servern und -Clients eingegangen.

Anschließend werden die Möglichkeiten zur Leistungsverbesserung und Reduzierung der Netzwerklast durch die Erweiterung der Squid-Software durch sogenannte Redirector vorgestellt und diesbezüglich durchgeführte Analysen ausgewertet.

Abschließend werden Möglichkeiten zur Beobachtung der aktuellen Status-Daten von Squid diskutiert und Empfehlungen für die Beobachtung besonders wichtiger Status-Daten gegeben, aus denen sich Engpässe bei der Diensterbringung bzw. Ursachen für im Betrieb aufgetretene Störungen ableiten lassen.

3.1 Konfiguration und Anforderungen von Squid

Im folgenden wird auf die wichtigsten, durch den Langzeitbetrieb von Squid in der Version 1.1.x gesammelte Erfahrungen bzgl. der Proxycaches-Software und deren wichtigsten Parameter selbst sowie Solaris 2.5.1 eingegangen werden. Ziel dabei ist es, Empfehlungen für Parameter zugeben, die großen Einfluß auf die Leistungsfähigkeit (Performance) sowie Betriebsbereitschaft von Squid haben.

In vielen Fällen wird dazu der betreffende **Parameter**, in der Konfigurationsdatei *squid.conf* von Squid als *TAG* bezeichnet, konkret benannt. Die für die Leistungsfähigkeit von Squid weniger ausschlaggebenden, zahlreichen TAGs werden im Rahmen dieser Arbeit nicht betrachtet. Besteht daran jedoch ein Interesse, können Erläuterungen dazu der Squid home page (http://squid.nlanr.net) bzw. der im Softwarepaket enthaltenen Standardkonfigurationsdatei entnommen werden.

3.1.1 Festplatten-Speicher (Cache Area)

Squid erlaubt die Konfiguration beliebig vieler Verzeichnisse (TAG *cache_dir*), unter der die zu cachenden Objekte gespeichert werden. Aus Gründen der Effizienz werden in jedem Cache-Verzeichnis per default 16 Unterverzeichnisse (als Level1-Verzeichnisse bezeichnet) und in selbigen jeweils 256 weitere Unterverzeichnisse (sogenannte Level2-Verzeichnisse) bei dem erstmaligen Start von Squid angelegt, in denen jeweils maximal 256 Objekte gespeichert werden (siehe Abbildung 3.1). Generell gehen die Entwickler der Software davon aus, daß sich mehr als 256 Unterverzeichnisse bzw. Dateien in einem Unterverzeichnisse negativ auf die Performance bei Einlesen der entsprechenden Einträge auswirkt.

Geht man von einer durchschnittlichen Objektgröße von 20 KB aus (Erfahrungswerte von proxycache.cs, wo Objekte bis max. 12.5 MB gecachet werden), können damit theoretisch 1 M (d.h. 16 * 256 * 256) Objekte mit einer Gesamtgröße von 20 GB in einem Cache-Verzeichnis gespeichert werden.

Squid versucht, alle zu speichernden Objekte möglichst gleichmäßig auf alle vorhandenen Level2-Verzeichnisse bzgl. bereits vorhandener Einträge (d.h., nicht nach dem noch zur Verfügung stehenden oder bereits genutzten Plattenplatz) zu verteilen. Dies trifft auch zu, wenn mehrere Cache-Verzeichnisse (cache_dir) vorhanden sind. Damit erklärt sich auch die etwas unterschiedliche Auslastung der verschiedenen Verzeichnisse bzgl. des benutzten Plattenplatzes.

Vom Proxycache.cs werden drei Cache-Verzeichnisse auf drei verschiedenen Partitionen zu je 8.3 GByte benutzt, wobei die Differenz bzgl. des belegten Plattenplatzes innerhalb eines Levels i.d.R. wie folgt aussieht:

- Cache-Verzeichnisse: ca. 10 ... 150 MB
- Level1-Verzeichnisse: ca. 1 ... 90 MB
- Level2-Verzeichnisse: ca. 0 ... 10 MB

Kann Squid aufgrund von Speichermangel auf der Partition mit dem entsprechenden Cache-Verzeichnis ein Objekt nicht mehr speichern, wird automatisch die Routine zur Löschung von derzeit am wenigsten genutzten Objekten (least recently used ... LRU) gestartet, was i.d.R. auch zur unnötigen Löschung von Objekten aus den anderen Cache-Verzeichnissen führt. Daraus folgt auch, daß die Auslastung aller Cache-Verzeich-

nisse bzgl. Plattenplatz von der Größe der kleinsten Partition mit einem oder mehr Cache-Verzeichnissen abhängig ist. Dies muß bei der Festlegung der Cache-Verzeichnisse berücksichtigt werden, da bei einer unbedachten Partitionierung Plattenplatz ungenutzt bleibt.

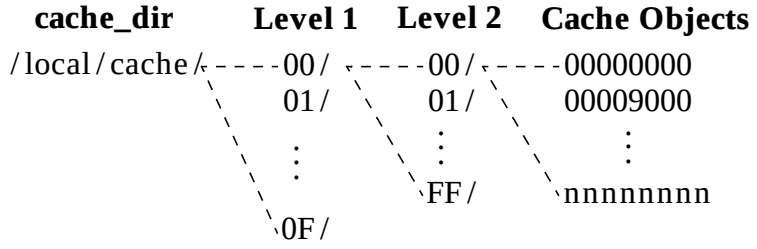

Abbildung 3.1: Schematische Darstellung der Cache-Verzeichnis-Struktur

Deshalb ist es ratsam, für jedes Cache-Verzeichnis eine separate Partition (i.w. als Cache-Verzeichnis-Partition bezeichnet) mit möglichst gleicher Größe anzulegen, um eine bestmögliche Auslastung selbiger zu erreichen[1]. Um die Performance von Squid bzgl. der Zeit beim Lesen von Objekten von der Platte zu steigern, wird empfohlen, wenn möglich alle Partitionen auf unterschiedlichen Festplatten anzulegen. Weitere Verbesserungen können i.d.R. erreicht werden, indem die benutzten Platten an verschiedene SCSI-Controler angeschlossen werden, da damit mehr Bandbreite zwischen Controler und angeschlossenen Platten zur Verfügung steht und die Controler selbst entlastet werden.

Da Squid bereits zu speichernde Objekte gleichmäßig über alle Level2-Verzeichnisse und somit Platten verteilt, scheint die Nutzung eines RAID-Feldes mit Level 0 (stripes) unnötig und könnte sich sogar negativ auf die Gesamt-Performance von Squid bei Verwendung von software-gesteuerten RAID-Feldern auswirken, da dann wiederum Software versucht, was Squid eigentlich schon leistet (nämlich die gleichmäßige Verteilung der Dateien auf verschiedene Platten) und damit wiederum Verzögerungen beim Schreiben bzw. Lesen entstehen. Der Beweis dafür muß aber aufgrund fehlender Hardware an dieser Stelle schuldig geblieben werden.

1. Ab Squid in der Version >= 1.2 werden Cache-Verzeichnis-Partitionen mit unterschiedlicher Größe unterstützt und können somit bzgl. vorhandenem Platz besser ausgenutzt werden. Damit besteht dahingehend nicht mehr die Notwendigkeit für möglichst gleich große Partitionen.

Ein weiterer wichtiger Punkt bei der Konfiguration der Parameter hinsichtlich des Festplattenspeichers stellt die Anzahl der Level1- und Level2-Verzeichnisse dar. Man sollte sich von Anfang an klar darüber sein, wie groß eine Cache-Verzeichnis-Partition in ferner Zukunft maximal werden kann, um so von vornherein die notwendigen Anzahl der Level1- und Level2-Verzeichnisse richtig per Konfigurationsdatei (squid.conf: TAGs *swap_level1_dirs, swap_level2_dirs*) festlegen zu können. Werden diese Parameter im nachhinein geändert, erstellt Squid die gesamte Cache-Verzeichnis-Struktur neu und der bisherige Inhalt geht verloren[1]! Für die Berechnung der Größe dieser Parameter können z.B. Gleichung 3.1 und Gleichung 3.2 genutzt werden, wobei angemerkt werden muß, daß die durchschnittliche Größe der gespeicherten Objekte auch von der maximalen Größe zwischenspeicherbarer Objekte (TAG *maximum_object_size*) abhängig ist.

Gleichung 3.1:
$$S_d = L_1 \times L_2 \times 256 \times S_o$$

Gleichung 3.2:
$$L_2 = \frac{S_d}{L_1 \times 256 \times S_o}$$

mit:

S_d	...	Größe der Cache-Verzeichnis-Partition
S_o	...	angenommene durchschnittliche Größe der gespeicherten Objekte
L_1	...	Anzahl der Level1-Verzeichnisse (mit $1 \le L_1 \le 256$)
L_2	...	Anzahl der Level2-Verzeichnisse (mit $1 \le L_2 \le 256$)

Des weiteren muß die Anzahl der benutzten Cache-Verzeichnisse mit Sorgfalt gewählt werden. Wird diese später geändert, geht wiederum der gesamte Cache-Inhalt verloren, da die Verzeichnisstruktur komplett neu aufgebaut wird[2].

Proxycache.cs hat seit seiner Inbetriebnahme drei Cache-Verzeichnisse, was sich bewährt hat, da so bereits zwei Mal die Kapazität der Cache-Verzeichnis-Partitionen

1. Für Squid in der Version >= 1.2 dürfte gleiches zutreffen, mit der Ausnahme, daß nur das betreffende Cache-Verzeichnis und nicht alle vorhandenen zerstört und neu aufgebaut werden.
2. Ab Squid Version >= 1.2 wird es möglich sein, neue Cache-Verzeichnisse nachträglich zu den bereits bestehenden hinzuzufügen, ohne daß der bisherige Inhalt aller Cache-Verzeichnisse verloren geht.

ohne Probleme mittels geeigneter Techniken erweitert werden konnte, ohne daß der Cache-Inhalt verloren ging[1] (siehe Tabelle 3.1):

Datum	Bemerkung	Platten	Partitionen
September 1996	Inbetriebnahme	2 GB: 1 x ST32430N (Fast SCSI II) 4 GB: 1 x ST15150 N (Fast SCSI II)	1 x 2 GB 2 x 2 GB
August 1997	Aufrüstung auf 12 GB + 1 x UWD-Controler	4 GB: 1 x ST15150 N (Fast SCSI II) 8 GB: 1 x ST19101W (UWD-SCSI)	1 x 4 GB 2 x 4GB
Juni 1998	Aufrüstung auf 24 GB + 1 x WD-Controler	8 GB: 1 x ST19101W 8 GB: 2 x ST410800W (WD-SCSI)	1 x 8 GB 2 x 8 GB

Tabelle 3.1: Aufrüstung und Cache-Verzeichnis-Größen von proxycache.cs

Die für die Cache-Verzeichnisse nicht mehr notwendigen Platten werden ab dem Zeitpunkt der jeweiligen Aufrüstung zur Speicherung der verschiedenen, von Squid generierten Logdateien benutzt, um später auf deren Basis Analysen durchführen zu können. Generell ist aber die Protokollierung der Zugriffe (access_log), der Software-Version der Clients (agent_log) sowie der Speicheroperationen bzgl. Festplatten (store_log) nicht notwendig und können deshalb abgeschaltet werden, falls sie nicht zu weiteren Untersuchungen (z.B. Ermittlung von HIT- und MISS-Raten, Transfervolumen etc.) benötigt werden. Das Abschalten der entsprechenden Logdateien kann sogar eine minimale Verbesserung der Dienstgüte verursachen, da weniger Festplattenzugriffe notwendig sind. Werden Logdateien geführt, sollte deren Größe regelmäßig überprüft und ggf. eine Rotation der Logdateien (*squid -k rotate*) herbeigeführt und die älteste der Logdateien gelöscht oder verschoben werden, denn kann Squid aufgrund voller Festplatten nicht mehr in die Logdateien schreiben, sinkt dessen Performance sehr sehr stark ab!!!

3.1.2 Hauptspeicher

Einer der grundlegendsten Bedingungen für den reibungslosen Betrieb von Squid ist der verfügbare Hauptspeicher (RAM). Um eine möglichst hohe Performance zu erreichen, werden von Squid sehr viele Daten im RAM gehalten. Dazu zählen u.a.:

- Platten-Schreib- und Lesepuffer (Disk I/O buffer)
- Netzwerk-Schreib- und Lesepuffer (Network I/O buffer)

1. Für das Verschieben des alten Cache-Inhalts auf eine neue Partition kann erfahrungsgemäß als Richtwert eine Transferrate von ca. 1 GB/h angenommen werden.

- IP-Adressen-Cache[1] (notwendig, da die Auflösung von FQDNs[2], auch als symbolische IP-Adressen bezeichnet, i.d.R. relativ zeitaufwendig ist)
- FQDN Cache (wenn dieser per Konfiguration explizit eingeschaltet ist)
- Netdb ICMP measurement database (RTTs zu Neighbors und WWW-Servern)
- Status-Informationen für aktuelle Anfragen, die u.a. den kompletten Header der Anfrage als auch den Header der Antwort enthalten
- verschiedene statistische Informationen
- „Hot Objects“ und „in-transit“- Objekte, d.h. Objekte, die sehr oft abgefragt werden bzw. die gerade von einem Neighbor oder WWW-Server heruntergeladen oder mind. an einen Client gesendet werden (in-transit objects)
- Objekt-Metadaten (Status-Informationen zu jedem gecacheten Objekt bzw. „hot object“, sowie Informationen zu dessen Identifizierung und Auffinden in den Cache-Verzeichnissen)

Der meiste RAM wird dabei durch die „hot objects“ und Objekt-Metadaten belegt. Die Objekt-Metadaten werden in einer Struktur gespeichert, die 52 Byte groß ist. Dazu kommt noch die damit assoziierte URL-Zeichenkette, die wie schon mehrmals angedeutet, erfahrungsgemäß ca. 50 Zeichen also ca. 51 Bytes groß ist. Somit werden pro Cache-Objekt ca. 103 Bytes im Speicher gehalten, was bei einer durchschnittlichen Objektgröße von ca. 13 KB und einem Cache Area von 1 GB ca. 7.9 MB (103 MB / 13) bzw. bei einer durchschnittlichen Objektgröße von ca. 20 KB ca. 5.1 MB (103 MB / 20) bei voll ausgelastetem Cache Area ausmachen würde. Da der proxycache.cs über ein Cache Area von 24 GB verfügt und die durchschnittliche Größe der zwischengespeicherten Objekte ca. 20 KB beträgt, benötigt dieser somit allein für die Objekt-Metadaten ca. 124 MB RAM! Dazu kommen erfahrungsgemäß ca. 30% des tatsächlich durch die Objekt-Metadaten genutzten Speichers für alle anderen oben aufgezählten Daten exklusive „hot objects“ (i.w. als andere Daten bezeichnet) , womit man beim proxycache.cs bereits bei einen RAM-Bedarf von ca. 161 MB RAM angelangt. Werden dazu noch ca. 20 MB vom Betriebssystem Solaris 2.5.1 und standardmäßig laufender Prozesse benötigter Speicher addiert, resultiert dies in einer Minimal-Anforderung an RAM von bereits 181 MB!

1. IP-Cache-Größe kann in der Squid-Konfigurationsdatei festgelegt werden (TAG *ipcache_size*)
2. FQDN ... Fully Qualified Domain Name (z.B. proxycache.cs.uni-magdeburg.de)

Da Squid für alle Objekte, die im Augenblick empfangen oder gesendet werden (sogenannte „in-transit objects“) ebenfalls Speicher benötigt, muß dieser ebenfalls zum RAM-Bedarf des Caches hinzugefügt werden. Dabei ist zu berücksichtigen, daß ein Objekt grundsätzlich vollständig in den Speicher geladen wird (egal ob es von dem lokalen Cache Area oder einem Neighbor / WWW- oder FTP-Server geladen wird), bevor es verworfen bzw. mit einmal in einem entsprechenden Level2-Verzeichnis abgespeichert wird. Ausnahme hierbei bilden Objekte, die größer als die vorgegebene max. Cache-Objekt-Größe sind (TAG *maximum_object_size*). Stellt Squid fest, daß ein „in-transit“ befindliches Objekt diese Größe übersteigt, werden alle Daten des Objektes, die bereits an den Client gesendet wurden verworfen und der allokierte Speicher für andere Objekte freigegeben. Wird also z.B. ein Objekt mit einer Größe von 12 MB von einem relativ „langsamen“ Server mit durchschnittlich 4 KB / s geladen, dauert die gesamte Übertragung ca. 51 Minuten. D.h. nach 25 Minuten werden bereits 6 MB RAM belegt, die mindestens noch weitere 26 Minuten dem Cache für andere Objekte nicht zur Verfügung stehen. Damit dürfte klar sein, daß der Speicher für „in-transit objects“ (TAG *cache_mem*) je nach Anzahl der gleichzeitig auf einen Cache zugreifenden Clients sowie den durchschnittlichen Transferraten zu den Neighbors bzw. Originalservern relativ hoch konfiguriert werden muß, um eine hohe Performance des Caches garantieren zu können. Steht wenig RAM zur Verfügung, kann die Reduzierung der *maximum_object_size* zur Minderung des Problems beitragen, aber es sollte dann klar sein, daß sich dies auch negativ auf die Byte-Hitrate des Proxycaches auswirken wird, da nur noch relativ kleine Objekte zwischengespeichert werden.

Weiterhin wird die unter *cache_mem* angegebene RAM-Größe auch für das negative Cachen von Objekten (d.h., fehlgeschlagene Anfragen werden für eine bestimmte Zeit, per default 5 min, ebenfalls gecachet und bei weiteren Anfragen innerhalb der folgenden 5 min sofort an den Client geschickt) genutzt. Der noch freie, nicht durch „in-transit“ und „negative cached“ objects belegte Speicher wird für sogenannte „hot objects“ (Objekte die sehr häufig abgefragt werden) genutzt.

Somit steht fest, daß auch dem Parameter *cache_mem* besondere Aufmerksamkeit bei der Konfiguration verdient. Generell sollte man diesen Wert so hoch wie möglich setzen, aber wie schon oben erwähnt, sollte dies auch in Abhängigkeit von der Anzahl der gleichzeitig zugreifenden Clients und zur Verfügung stehenden Bandbreite festgelegt

werden! Wird dieser Wert allerdings zu hoch gesetzt, kann dies bewirken, daß Squid virtuellen Speicher auf der Festplatte benutzen muß, d.h. Daten aus dem RAM-Speicher auf Festplatte ausgelagert und später wieder eingelesen werden müssen (swapping) und damit die Performance der Applikation deutlich sinkt. Geht man systematisch vor, könnte dieser Wert durch ständiges Monitoring der Cache Response Time sowie unter Zuhilfenahme der durch das Betriebssystem zur Verfügung gestellten Tools zum Beobachten der Swap- und Disk-Aktivitäten (bei Solaris 2.x z.B. mittels *iostat -xtc*, *vmstat* oder *swap -s*) in die Nähe des Optimums geführt werden.

Als das Proxycache-Projekt an der Universität Magdeburg gestartet wurde, besaß die ursprünglich verwendete Maschine nur 64 MB RAM. Dies führte sehr schnell zu den oben beschriebenen Hauptspeicher-Engpässen und somit zu einer relativ langsamen Diensterbringung. Deshalb wurde innerhalb von einer Woche eine weitere Workstation und ca. 6 Monate später eine dritte Workstation in den Magdeburger Proxycache-Verbund integriert, und die IP-Subnetze der Universität bzgl. des zuerwartenden Verkehrs gleichmäßig auf selbige aufgeteilt, um einen entsprechend schnellen Dienst garantieren zu können. Nachdem der Parent der MCH (proxycache.cs) auf vorerst ausreichend RAM (384 MB) aufgerüstet werden konnte und der Proxycache des URZ mit ebenfalls 384 MB in Betrieb ging, wurde konsequenter Weise der Proxycache-Dienst der zwei Maschinen mit nur 64 MB RAM eingestellt und somit auch zwei eventuelle Fehlerquellen „Points-of-Failure“ in der MCH beseitigt.

Für den proxycache.cs werden derzeit 256 MB als cache_mem benutzt, was final (unter Berücksichtigung des 24 GB großen Cache Area und anderer Daten) zu einem RAM-Bedarf von ca. 437 MB führt. Da diese Maschine 512 MB RAM besitzt, steht auch ein gewisser Puffer bereit, so daß das Swappen i.d.R. vermieden wird.

3.1.3 Feinabstimmung des Betriebssystems

Ein weiterer, für den Proxycache-Dienst essentieller Parameter, ist die für einen Prozeß zur Verfügung stehende Anzahl von Filedescriptors, die gewöhnlich durch das verwendete Betriebssystem begrenzt sind. Standardmäßig ist diese z.B. bei Solaris 2.x 1024, bei FreeBSD und Linux 256. Da für jede Verbindung (jeden socket) als auch jede geöffnete Datei jeweils ein Filedescriptor benötigt wird, kann eine zu niedrige Anzahl von verfügbaren Filedescriptors zu einem weiteren Engpaß führen. D.h. es können keine neuen Filedescriptors angelegt werden, und damit weder neuen Verbindungen

zu Neighbors oder WWW-Servern aufgebaut, noch zwischengespeicherten Dateien geöffnet werden, womit praktisch zu diesem Zeitpunkt der Proxycache-Dienst nur noch eingeschränkt nutzbar ist.

Somit muß in Abhängigkeit von der zu erwartenden Anzahl von Anfragen an den Proxycache versucht werden, diesen durch das verwendete Betriebssystem limitierten Parameter zu erhöhen.

Unter Solaris >= 2.4 ist dies zu bewerkstelligen, indem in der Datei */etc/system* der entsprechende Parameter eingetragen wird (*set rlim_fd_max = 4096*), welcher bei einem Neustart der Maschine aktiviert wird. Eine Applikation kann dann unter Nutzung des Systemaufrufs *setrlimit()* die Anzahl der zur Verfügung stehenden Filedescriptors bis zur eingetragenen Anzahl erhöhen.

Zu beachten ist, daß Squid bei Erhöhung von *rlim_fd_max* auf > 1024 unter Solaris mit der Option *--enable-poll* kompiliert werden muß, da ansonsten der Systemaufruf *select()* benutzt wird, der nicht mehr als 1024 Filedescriptors unterstützt. Durch Nutzung der oben aufgeführten Option nutzt Squid statt *select()* den Systemaufruf *poll()*, der keine Grenze hinsichtlich benutzbarer Filedescriptors unterliegt, um herauszufinden, ob neue Daten vom einem Filedescriptor gelesen werden können.

Für andere Betriebssysteme wie Linux und FreeBSD gibt es ebenfalls entsprechende Methoden zur Erhöhung der verfügbaren Filedescriptors je Prozeß (i.d.R. Neukompilierung des Kernels mit entsprechenden Parametern), welche in der Mailing List squid-users@ircache.net erfragt oder in dessen Archiv (http://squid.nlanr.net/Mail-Archive/squid-users/hypermail.html) nachgelesen werden können.

Weitere wichtige Hinweise für die Feinabstimmung (Tuning) von Solaris 2.x bzgl. Squid sind speziell in [31] und [32], sowie im allgemeinen in [33] - [36] enthalten.

3.2 Konsistenz von Cache-Objekten

Eine der wichtigsten Kriterien für die Akzeptanz von Proxycaches ist, daß diese die Aktualität der zwischengespeicherten Objekte gewährleisten. Das Ergebnis einer Anfrage an einen Proxycache darf sich nicht von dem der Anfrage an den Originalserver für das entsprechende Objekt unterscheiden. Wird dies nicht gewährleistet, kann es passieren, daß der Nutzer eines Proxycaches teilweise veraltete oder auf dem Originalserver nicht mehr vorhandene Objekte geliefert bekommt, was sich gerade bei sich re-

gelmäßig ändernden Seiten (wie z.B. die einer Tageszeitung oder Seiten, die das aktuelle Fernsehprogramm enthalten) sehr negativ auf die Akzeptanz des Dienstes auswirkt.

3.2.1 Squids Regeln für das Cachen von Objekten

Um eine möglichst hohe Konsistenz der zwischengespeicherten Objekte mit den Originalen auf den entsprechenden Servern, aber trotzdem einen relativ hohen Nutzungsgrad[1] zu erreichen, nutzt Squid einen relativ ausgefeilten Algorithmus, der in Abbildung 3.2 dargestellt ist. Die darin verwendeten Parameter sind in Tabelle 3.2 erläutert. Dabei wird der Term „*fresh*“ verwendet, wenn das jeweilige Objekt (immer noch) als aktuell betrachtet und entsprechend weiterverarbeitet, d.h. an den anfragenden Client ausgeliefert werden kann. Der Term „*stale*“ wird dagegen verwendet, wenn das Objekt als veraltet behandelt und somit durch den Proxycache erneut validiert bzw. vom Originalserver heruntergeladen werden muß. Der Zustand, ob ein Objekt fresh oder stale ist, wird als *Freshness* bezeichnet.

check_time	Zeitpunkt, zu dem die Prüfung erfolgt
r_max_age	Das im Object Request Header angegebene Alter (Max-Age), welches das angeforderte Objekt maximal haben darf.
e_expires	Zeitpunkt, ab dem das entsprechende Objekt nicht mehr als frisch („fresh“), sondern nur noch als veraltet („stale“) verarbeitet werden darf (-1, falls das entsprechende Objekt kein *Expires* Feld im Header enthält).
e_lm	Zeitpunkt, zu dem das Objekt das letzte Mal modifiziert wurde (-1, falls das Objekt kein *Last-Modified* Feld im Header enthält)
e_timestamp	Zeitpunkt, zu dem das Objekt vom Originalserver heruntergeladen bzw. das letzte Mal revalidiert wurde.
age	Alter eines zwischengespeicherten Objektes, d.h. Zeitspanne zwischen *e_timestamp* und *check_time*
min	minimales Alter eines zwischengespeicherten Objektes (konfigurierbar)
max	maximales Alter eines zwischengespeicherten Objektes (konfigurierbar)
percent	Verhältnis aus *age* und der Zeitspanne zwischen *e_lm* und *e_timestamp*

Tabelle 3.2: Parameter für die Überprüfung der Freshness eine Objekts

1. derzeit wird davon ausgegangen, daß i.d.R. bestenfalls ca. 50-70% aller Client Anfragen durch Cache-HITs abgedeckt werden können (siehe [37], [38], [39] und [40]).

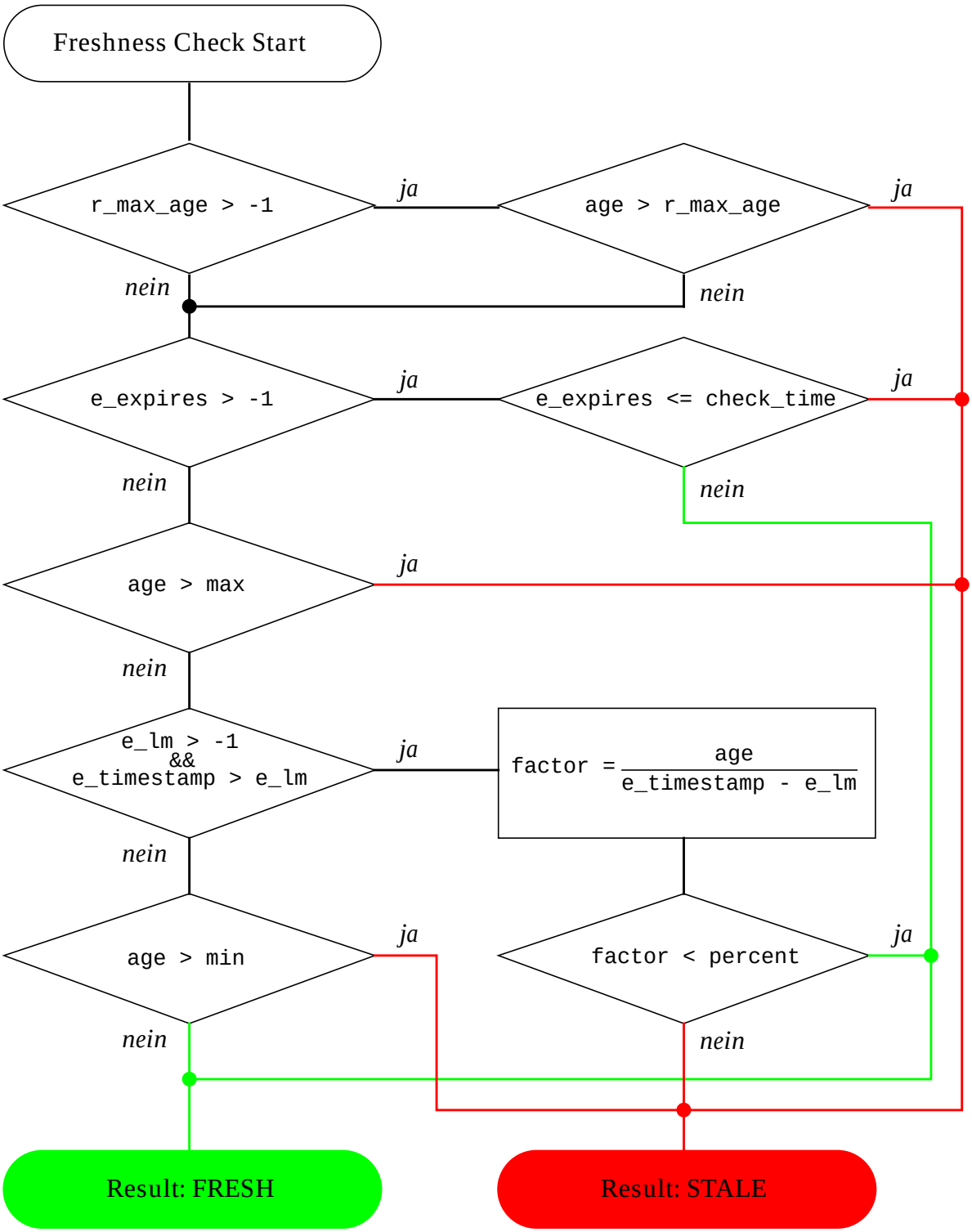

Abbildung 3.2: Flußdiagramm zur Überprüfung der Freshness eines Objekts

Der in Abbildung 3.2 gezeigte Algorithmus entspricht dem in der Squid Version >= 1.2b. Die Proxycaches proxycache1.cs und proxycache2.cs verwenden derzeit zwar

noch Squid 1.1.22, doch wurde vor der Kompilierung der Quellcode für diese Routine ausgewechselt, da der Algorithmus in Version 1.1.x u.U. die Freshness von Objekten nicht korrekt bestimmt (d.h., hier ist ein Objekt immer fresh, wenn dessen Alter niedriger als das vorgegebene Minimum ist: *age* < *min* => *fresh*)[1].

Squid erlaubt es, unter Zuhilfenahme von regulären Ausdrücken die Parameter *min, percent, max* für entsprechende Objekte in der Squid-Konfigurationsdatei mittels dem TAG *refresh_pattern* zu setzen. Standardmäßig verwendet Squid die folgende Einstellung:

TAG	regulärer Ausdruck	min	percent	max	options
refresh_pattern	.	0	20%	4320	

Der reguläre Ausdruck wird mit dem URL des Objekts verglichen, und falls dieser zu dem URL paßt (pattern match), die angegebenen Parameter verwendet. Da ein '.' zu jedem URL paßt, gelten standardmäßig die Werte *min* = 0 Minuten, *percent* = 20% und *max* = 4320 Minuten = 72 Stunden = 3 Tage).

Dies sind natürlich sehr konservative Werte. Gesammelte Erfahrungen zeigen z.B., daß sich die URLs für Bilder, Grafiken, Sound und Animationen bzgl. ihres Inhalts i.d.R. nur sehr sehr selten ändern, so daß hier die Parameter ohne große Bedenken je nach Größe des zur Verfügung stehenden Cache Area z.B. auf *min = 10080* (= 7 Tage), *percent = 100%* und *max = 40320* (= 30 Tage) gesetzt und somit bessere HIT-Raten erzielt werden können.

Ähnliches gilt auch für FTP- und Gopher-Objekte, denn auch hier ändert sich i.d.R. der Name einer Datei und somit deren URL nur, wenn sich der Inhalt selbiger auch ändert (z.B. Versionsnummer).

Beim Betrieb von proxycache.cs und proxycache1.cs konnte durch die Ausnutzung von *refresh_pattern*s die HIT-Rate um ca. 5-10% gesteigert werden. Die dabei verwendete Konfiguration kann dem Anhang B.1 entnommen werden.

---

1. In Squid Version >= 1.2b kann immer noch die Regel *age* < *min* => *fresh* aktiviert werden, allerdings müssen dazu die Squid-Quellcodes mit der Option **HTTP_VIOLATIONS** kompiliert und in der Konfigurationsdatei die Optionen **override-expire** und **override-lastmod** bei den jeweiligen *refresh_pattern*s gesetzt werden!

3.2.2 Einfluß von WWW-Servern

Die Analyse des von Squid verwendeten Algorithmus zur Überprüfung der Freshness von Objekten als auch das Studium des RFC zum HTTP/1.1 [9] führt zu weiteren Erkenntnissen bzgl. der Verwendung und Konfiguration von WWW-Servern und der Gestaltung von eigenen WWW-Seiten.

So sollte z.B. grundsätzlich ein WWW-Server verwendet werden, der bei jedem ausgelieferten Objekt einen Header mit einem gültigen Last-Modified bzw. Expire-Feld generiert, um ein korrektes Zwischenspeichern durch Caches zu ermöglichen und die Vorraussetzungen dafür zu schaffen, daß die zwischengespeicherten Objekte möglichst konsistent mit denen auf dem WWW-Server sind.

Ebenso erlauben moderne WWW-Server wie z.B. Apache[1], einer der weltweit am meisten eingesetzten WWW-Server[2], spezielle Konfigurationsdateien (i.d.R. *.htaccess*) auf Basis von Verzeichnissen, die den Server instruieren können, für bestimmte Dateien spezifizierte Werte in den Expire-Feldern einzutragen, falls diese Dateien abgefragt werden. Nützlich ist dies beispielsweise bei Bildern, die regelmäßig ihren Inhalt, nicht aber ihren Namen ändern. Ein praktisches Beispiel dafür sind die für den Parent der MCH periodisch erstellten Statistiken[3], die eine Generierung neuer Grafiken mit gleichem Namen alle 5 min bewirken. Da bei einigen Grafiken die jeweiligen Änderungen alle 5 min nur minimal und kaum sichtbar sind, kann deren Expire-Feld mit einem entsprechend größeren Wert versehen und damit ein längeres Cachen selbiger bewirkt werden. Auf diesen Überlegungen basierend wurde die nachfolgend aufgelistete *.htaccess* Datei angelegt, um so die Voraussetzung für die Konsistenz der Objekte in Caches und auf dem WWW-Server zu schaffen. Weitere Einzelheiten zur Nutzung dieser Direktiven des Apache-WWW-Servers können unter http://www.apache.org/docs/mod/mod_expires.html studiert werden.

```
<Files „*-day.gif“>
ExpiresActive On
# 30 minutes = ca. 6 pixel
ExpiresDefault M1800
</Files>
```

1. siehe http://www.apache.org
2. siehe http://www.netcraft.com/survey/
3. siehe http://ivs.cs.uni-magdeburg.de/~elkner/stats/proxy/cache/

```
<Files „*-week.gif“>
ExpiresActive On
# 3 hours = ca. 6 pixel
ExpiresDefault M10800
</Files>

<Files „*-month.gif“>
ExpiresActive On
# 12 hours = ca. 6 pixel
ExpiresDefault M43200
</Files>

<Files „*-year.gif“>
ExpiresActive On
# 6 days = ca. 6 pixel
ExpiresDefault M518400
</Files>

<Files „*.html“>
ExpiresActive On
# 5 minutes =
ExpiresDefault M300
</Files>

# on this server ExpiresActive defaults to Off
#<Files „index.html“>
#ExpiresActive Off
#</Files>
```

Den Autoren von WWW-Seiten steht ein weiteres Mittel zur Verfügung, wenn der entsprechende WWW-Server für diese Seiten das HTML HEAD Element *META http-equiv* (siehe [41] und [42]) honoriert. Damit kann der WWW-Server instruiert werden, gezielt Felder im Response Header für ein abgefragtes Objekt zu erzeugen. Ein Beispiel dafür wäre das Element:

```
<META http-equiv=“Cache-Control“ content=“proxy-revalidate“>
```

woraus der WWW-Server folgendes Feld im Response Header eintragen würde:

```
Cache-Control: proxy-revalidate
```

Damit muß ein Proxycache bei jeder Anfrage nach dem entsprechenden Objekt die Äquivalenz des zwischengespeicherten Objektes mit dem originalen Objekt überprüfen. Wie das dann der Proxycache bewerkstelligt, ob er das Objekt wiederholt vollständig vom Originalserver herunterlädt oder mittels HEAD Request nur die Übereinstimmung der Header des zwischengespeicherten und originalen Objekts überprüft und daraus über die Freshness des zwischengespeicherten Objektes ent-

scheidet, kann weder durch den WWW-Server noch den User Agent (also i.d.R. dem WWW-Browser) beeinflußt werden. Allerdings wäre in diesem Fall die Unterstützung von HEAD Requests seitens des WWW-Servers wünschenswert, was aber erst ab HTTP/1.1 [9] gefordert wird, damit die Übertragung des HTML-Bodies vermieden, somit bei vorhandener Äquivalenz Bandbreite eingespart und die Anfrage schneller beantwortet werden kann.

3.2.3 Einfluß von User Agents

Auch User Agents (Programme, die Anfragen an Proxycaches stellen), können Proxycaches durch gezielte Anfragen instruieren, zwischengespeicherte Objekte hinsichtlich ihrer Aktualität zu überprüfen bzw. neu vom Originalserver herunterzuladen und somit die Konsistenz zwischengespeicherter Objekte beeinflussen.

Speziell moderne WWW-Browser bieten die Möglichkeit, durch Auslösung eines Refresh/Reload-Befehls für das bereits geladene Dokument den Proxycache zu veranlassen, die Gültigkeit dessen auf dem Originalserver erneut zu überprüfen und ggf. neu herunterzuladen. Der Netscape Communicator alias Mozilla sowie der Microsoft Internet Explorer (die lt. eigenen Studien 80-90% aller WWW-Anfragen ausmachen; siehe Anhang A.5) bewerkstelligen dies, indem sie der Anfrage an den Proxycache ein If-Modified-Since als auch ein "Pragma: no-cache"-Feld hinzufügen. Pragma: no-cache bedeutet nicht, daß das betreffende Objekt nicht zwischengespeichert werden darf, sondern die Anfrage an den Originalserver weitergeleitet werden muß. Eigentlich überprüft also nicht der Proxycache die Gültigkeit des Objektes, sondern der WWW-Browser und befiehlt dem Proxycache, das Objekt neu vom Originalserver herunterzuladen, selbst wenn dieser bereits eine aktuelle Kopie davon besitzt.

Löst ein WWW-Browser also solch eine Refresh/Reload-Aktion aus, sendet er eine Anfrage mit den oben bereits erwähnten Feldern im Header selbiger. Der Proxycache leitet nun auf grund des "Pragma: no-cache"-Feldes diese Anfrage an den Originalserver weiter. Hat sich das Objekt auf dem Originalserver nicht seit dem im "If-Modified-Since"-Feld angegebenen Datum geändert, schickt dieser eine Antwort, die den Status 304 (not modified) enthält. Diese leitet der Proxycache an den WWW-Browser weiter, der damit auch weiß, daß das geladene Dokument noch immer aktuell ist. Stellt der Originalserver jedoch fest, daß sich das betreffende Objekt geändert hat, schickt er die-

ses komplett als Antwort an den Proxycache. Selbiger speichert das empfangene Objekt zwischen und leitet es unverändert an den WWW-Browser weiter.

Mozilla bietet darüber hinaus eine weitere Funktion, die jedoch überflüssig sein dürfte, wenn die Proxycaches der in HTTP/1.1 [9] geforderten Funktionalität der korrekten Behandlung von Anfragen mit "If-Modified-Since"-Feldern genügen und alle WWW-Server Anfragen mit "If-Modified-Since"-Felder korrekt beantworten würden (dem Autoren sind bisher keine solchen WWW-Server bekannt): Hält man in Mozilla die Shift-Taste gedrückt und klickt gleichzeitig mit der Maus auf den Reload-Knopf, wird eine Anfrage generiert, die kein "If-Modified-Since"-Feld, sondern nur ein "Pragma: no-cache"-Feld hinzufügt. Damit wird der Proxycache instruiert, das Objekt vom Originalserver herunterzuladen, egal ob es sich geändert hat oder nicht. Eine weitere Eigenheit (von sehr vielen Proxycache-Administratoren auch als Unsitte bzw. Bug bezeichnet) von Mozilla ist, daß alle Anfragen für Objekte, die aus der eigenen Bookmark-Liste aufgerufen werden, mit einem "Pragma: no-cache"-Feld versehen werden. D.h., der Proxycache wird wiederum instruiert, das entsprechende Objekt herunterzuladen, egal ob sich dies geändert hat oder nicht.

Da sehr viele Nutzer die von ihnen am meisten besuchten WWW-Seiten in ihrer Bookmark-Liste vermerkt haben und diese auch über selbige aufrufen, stellt dies eine unnötige, oftmals nicht zu vernachlässigende Last bzgl. involvierter Ressourcen (Bandbreite, Zeit, Server-Last) dar.

Deshalb ist auch der Wunsch vieler Squid-Administratoren zu verstehen, grundsätzlich "Pragma: no-cache"-Felder in Anfragen an den Proxycache in einen "If-Modified-Since"-Feld mit dem Datum des zwischengespeicherten Objektes (falls vorhanden) zu transformieren und entsprechend weiterzuverarbeiten. Dies wurde bei der Entwicklung von Squid berücksichtigt und ist ab Version 1.1.15 in Form des TAGs **reload_into_ims** konfigurierbar.

Abschließend bleibt zu sagen, daß es immer Software geben wird, die durch gewisse Optionen so konfiguriert werden kann, daß maximale HIT-Raten erreicht werden können, auch wenn dies zu einer Verletzung der Spezifikationen von RFC 2068 (HTTP/1.1) [9] führt. Deshalb finde ich persönlich die oben genannten Eigenheiten von Mozilla überflüssig und denke, daß dies in vielen Fällen sogar genau zum Gegenteil von dem führt, was damit wahrscheinlich angedacht ist oder sogar zu firmeninternen Re-

gelungen, diesen WWW-Browser nicht mehr benutzen zu dürfen. Hinreichend viele Beispiele existieren bereits für beide Aussagen (siehe u.a. http://squid.nlanr.net/Mail-Archive/squid-users/).

3.3 Redirector

Ein weiteres Mittel zur Verbesserung der Dienstqualität im WWW und Minderung von Kosten stellt im Zusammenhang mit Squid ein sogenannter *Redirector* dar: Squid kann so konfiguriert werden, daß er die Parameter jeder Anfrage (URL, IP-Adresse/FQDN des anfragenden Clients, dessen Identität sowie die Methode) an einen externen Redirector-Prozeß über die Standardaus- bzw. -eingabe übergibt, bevor er die Anfrage weiter verarbeitet. Dieser Redirector kann nun aufgrund bestimmter Regeln die übergebenen Parameter durch andere oder eine Leerzeile ersetzen und über die Standardausgabe an Squid zurückgeben. Wird eine Leerzeile zurückgegeben oder liefert der Redirector innerhalb einer bestimmten Zeit kein Ergebnis zurück, nutzt Squid die ursprünglichen Parameter, andernfalls die zurückgegebenen zur weiteren Verarbeitung. Redirectors sind nicht Bestandteil der Squid-Software. Sie müssen separat besorgt oder selbst geschrieben werden.

Damit ist es also möglich, Squid so zu konfigurieren, daß bestimmte WWW-Objekte von anderen, bzgl. Service schnelleren oder, und bzgl. Bandbreite kostengünstigeren Servern (z.B. lokalen FTP Mirrors) statt von dem Originalserver abgefragt bzw. durch vollständig andere WWW-Objekte ersetzt werden.

Natürlich spielt gerade bei FTP-Mirrorn die Freshness von Objekten eine besonders große Rolle, da das FTP-Protokoll über keine geeigneten Mechanismen verfügt, welche die Freshness von Objekten bestimmen lassen. Im Rahmen des „National Janet Web Cache Service“ der UK Academic and Research Community wurde deshalb ein Projekt namens „DOH - a JANET Web Cache Service/TERENA mirror tracking and URN seeding project“[1] gestartet, mit dem Ziel, Datenbanken über verschiedene FTP Mirrors und deren Inhalt zu erstellen und darüber hinaus ein automatisches Umleiten über die in Squid >= 1.2beta hinzugekommene URN Unterstützung (siehe [43]) unter Zuhilfenahme der erstellten Datenbanken) zu bewerkstelligen. Theoretisch sollte es demnach möglich sein, auch spezielle Redirectors zu implementieren, die ebenfalls diese Daten-

1. siehe http://wwwcache.ja.net/dev/doh/

banken nutzen können und so Squid 1.1.x bzw. Squid >= 1.2beta auch ohne installierte URN-Unterstützung sinnvolle Redirections zu ermöglichen.

Ein Beispiel für einen bei den Proxycaches der FIN eingesetzten Redirector ist *jesred* [30], der im Rahmen dieser Arbeit entwickelt wurde (ca. 2200 Zeilen Quellcode). Dieser wird aufgrund seiner hohen Geschwindigkeit (ca. 50.000 Anfragen je Sekunde[1]) als auch relativ hohen Flexibilität inzwischen nicht nur bei den proxycache.cs und proxycache1.cs erfolgreich genutzt, sondern auch weltweit von vielen anderen Squid-Proxycaches, sowohl im kommerziellen als auch nicht-kommerziellen Bereich.

Das primäre Ziel des Einsatzes von jesred auf proxycache.cs und proxycache1.cs ist derzeit, Anfragen für dynamische zu ladende Werbebilder/-banner und HTML-Seiten, die nur Werbung enthalten, durch ein leeres transparenteres Bild sowie eine „leere“ HTML-Seite zu ersetzen.

Das Problem dabei ist, daß inzwischen sehr viel Werbung auf verschiedensten WWW-Seiten existiert und viele Nutzer sich, ähnlich wie beim Fernsehen, dadurch belästigt fühlen. Ebenso müssen diese Werbebilder oftmals von stark überlasteten bzw. sehr langsamen WWW-Servern (wie z.B. ad.doubleclick.net, adsNN.focalink.com, adforce.imgis.com, u.v.a.m.) heruntergeladen werden. Desweiteren wird das Zwischenspeichern von Objekten durch Proxycaches von diesen Servern oftmals noch zusätzlich erschwert bzw. sogar absichtlich verhindert, indem:

- die jeweiligen URLs Parameter enthalten (Proxycaches speichern diese Objekte apriori nicht zwischen, da allgemein angenommen wird, daß es relativ unwahrscheinlich ist, daß mehrmals eine Anfrage bzgl. einem URL mit exakt den gleichen Parametern erfolgt. Diese Annahme ist bisher nicht bewiesen.) (z.B. adforce.imgis.com, ad.linkexchange.com, www.trafficcount.com)
- die Objekte ein „Pragma: no-cache“ bzw. ein „Cache Control“-Feld enthalten (z.B. ad.asv.de, adforce.imgis.com, ad.doubleclick.net - verursachte im Monat Oktober 1998 ca. 20% aller Redirections)
- die Objekte ein Expire-Feld mit einem Datum enthalten, daß der aktuellen Zeit bzw. einem weit zurückliegendem Datum entspricht (z.B. ad.asv.de - verursachte im Oktober 1998 ca. 10% aller Redirections, www.ad-server.de, adserv.spiegel.de)
- sich ständig ändernde Cookies (siehe [44]) verwendet werden (z.B. wNN.hitbox.com mit NN .. {01 ... 100})

1. zu diesem Test wurden die Redirect-Regeln aus http://ivs.cs.uni-magdeburg.de/~elkner/webtools/jesred/redirect.rules und als Eingabe 100.000 entsprechend modifizierte Einträge aus der „access_log“-Datei des proxycache.cs benutzt.

- oder nur einmal gültige Objekte „on the fly“ generiert (d.h. URL wird erst bei der Anfrage an den Server mit einem einmaligen Schlüssel erzeugt) und kurze Zeit später wieder gelöscht werden (z.B. www.heise.de; verursachte im November 1998 ca. 17% aller Redirections)

Fairer Weise muß hier allerdings bemerkt werden, daß ad.doubclick.net inzwischen auf ca. 75% Werbebilder-Anfragen mit einem WWW-Server-Redirect auf das eigentliche Werbebanner antwortet, welche aber keine der oben genannten Eigenschaften zur Verhinderung des Zwischenspeicherns benutzen.

Die Analyse der „rewrite_log“-Datei von proxycache.cs für den Monat Oktober 1998 hat ergeben, daß ca. 67% dieser Objekte nocheinmal abgefragt werden, was letztendlich zu einer HIT-Rate von 50% aller Anfragen bzgl. dieses Servers führen dürfte.

Desweiteren wurde ermittelt, daß jedes gültige Objekt (Antworten mit Code 200 inkl. aller Redirections) im Durchschnitt eine Größe von 6.5 KB hatte. Es wurden 63680 Objekte mit einem lokalen, 64 Byte großen Objekt (http://141.44.30.2/images/dot.gif) ersetzt, was eine theoretische Volumeneinsparung von ca. 410 MB bzgl. externem Verkehr in diesem Monat erzielt haben dürfte. Da ca. 20% aller Redirects via ad.doubleclick.net URLs erfolgten, und falls die Annahme über die oben errechnete 50% HIT-Rate stimmt, würde sich die Volumeneinsparung bzgl. des externen Verkehrs auf ca. 375 MB[1] reduzieren, wenn die entsprechenden URLs nicht mehr durch proxycache.cs auf eine URL für ein lokales Objekt umgeschrieben würden.

Letztendlich zeigt sich, daß bereits Redirects von WWW-Anfragen bzgl. Werbebanner zu einer Einsparung an Bandbreite und für den Download benötigter Zeit führen. Da die Analysen der Anfragen an proxycache.cs im folgenden Kapitel zeigen, daß volumenmäßig ca. 30% aller Objekte mittels FTP (d.h. schema ftp://) heruntergeladen werden, sollte in weiteren Arbeiten untersucht werden, inwieweit der Einsatz und die Nutzung lokaler oder im Breitband-Forschungsnetz des DFN (B-WiN) befindlichen FTP-Spiegel-Servern im Zusammenhang mit Proxycache-Redirects weitere Vorteile bzgl. Einsparung von Zeit beim Download und der Einsparung universitäts- bzw. B-WiN-externer Bandbreite führt. Dabei sollte ebenfalls ein Modell entwickelt und ge-

1. Bei der Berechnung wurde von einer durchschnittlichen Größe von 240 Byte einer Redirect-Antwort von ad.doubleclick.net ausgegangen (ermittelt bei der rewrite.log Analyse vom Monat Oktober 1998).

nutzt werden (evtl. auch das oben angesprochene DOH-Projekt integriert werden) , daß die Aktualität von FTP-Server-Spiegelungen überprüft und bei Inkonsistenzen mit dem Originalserver automatisch den Administrator des Proxycaches und evtl. auch gleich an den Administrator des Spiegelservers informiert.

3.4 Monitoring

Um einen möglichst störungsfreien, optimalen Proxycache-Dienst zu garantieren, ist es außerordentlich wichtig, diesen Dienst bzw. die entsprechende Maschine ständig zu beobachten (Monitoring), um ggf. Maßnahmen rechtzeitig einleiten zu können.

Da Squid eine Schnittstelle für Abfragen statistischer Daten (wie z.B. Anzahl derzeit genutzten Filedescriptors, aktuelle HIT- und MISS-Raten, transferiertes Volumen, LRU age, benutzter Hauptspeicher / Cache Area, Anzahl der Redirects, etc.) besitzt, ist dies durch eigene Programme oder durch den im Software-Paket enthaltenen Squid-Cache-Manager (i.w. SCM) möglich.

3.4.1 Der Squid-Cache-Manager

Der SCM ist ein CGI-Programm[1], welches eine relativ komfortable Abfrage aller aktuellen statistischen Daten eines Squid-Proxycaches über das WWW erlaubt und relativ übersichtliche Zusammenfassungen der erfaßten Daten generiert (siehe z.B. Abbildung 3.3 bzw. http://ivs.cs.uni-magdeburg.de/cgi-bin/squid.cgi). Er kann genutzt werden, um sich stichprobenartig einen relativ guten Überblick über den derzeitigen Zustand (Schnappschuß) eines Proxycaches zu verschaffen. Für ein dauerhaftes, kontinuierliches Monitoring eignet er sich jedoch nicht, da er immer nur aktuelle und Durchschnittswerte seit dem Programmstart der Proxycache-Software anzeigt. Es ist damit also nicht möglich, genauere Aussagen bzgl. der zu beobachtenden Parameter zu einem bestimmten, zurückliegenden Zeitpunkt zu erhalten, evtl. aufgetretene Störungen, zu knapp gewordene Ressourcen oder Überlastungen ausfindig zu machen und Analysen anzustellen. Aus diesem Grund ist es notwendig, andere Programme dafür zu nutzen.

1. siehe http://squid.nlanr.net/Squid/FAQ/FAQ-9.html

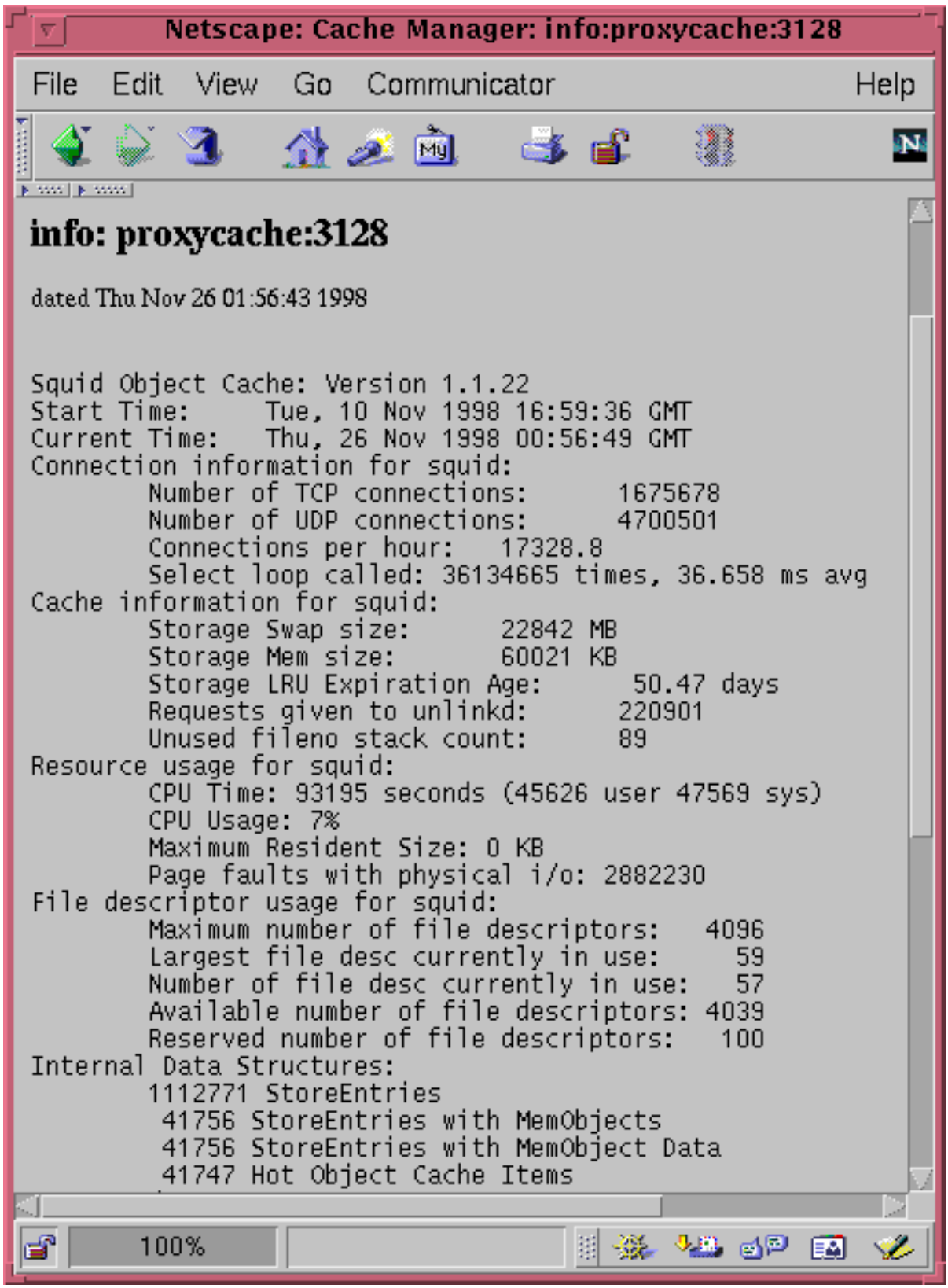

Abbildung 3.3: Squid Cache Manager: Allgemeine Informationen

3.4.2 Andere Monitor-Programme

Zum Monitoring der Proxycaches der MCH wurde aus oben genannten Gründen vom Autoren der Arbeit ein Proxycache-Monitoringsystem[1] (PCMS) entwickelt, welches die wichtigsten Statistiken jedes Proxycaches im Intervall von jeweils 5 Minuten abfragt, die erhaltenen Daten aufbereitet und als Grafik via WWW bereitstellt.

Um den Programmieraufwand möglichst gering zu halten, wurden zwei bereits vorhandene, via WWW frei verfügbare Tools sowie ein Erweiterungspaket für die Programmiersprache Perl in das selbst geschriebene Programm gps4a.pl (Umfang ca. 1000 Zeilen) integriert, welche in der Summe dann das PCMS ergeben:

echoping[a]	Programm, welches im Falle von PCMS die Zeit mißt, die zwischen dem Stellen einer Anfrage und dem Empfang der entsprechenden Antwort vergeht (Antwortzeit o. engl.: response time)
MRTG - Multi Router Traffic Grapher[b]	Tool zum Monitoring der Verkehrs-Last für verschiedene Netzwerk-Verbindung, welches HTML Seiten mit entsprechenden Grafiken im GIF-Format generiert und damit eine aktuelle (LIVE) Repräsentation der Netz-Verkehrs-Situation wiedergibt.
TimeDate-1.x[c]	Perl Erweiterung
gps4a.pl	Programm, welches über die Squid API gewünschte Daten vom Proxycache abfordert, mittels echoping die Antwortzeiten des o. der Proxycaches mißt, diese daten geeignet abspeichert und MRTG aufruft

Tabelle 3.3: Komponenten des PCMS

a. siehe ftp://ftp.internatif.org/pub/unix/echoping/
b. siehe http://ee-staff.ethz.ch/~oetiker/webtools/mrtg/mrtg.html
c. siehe ftp://ftp.uni-hamburg.de/pub/soft/lang/perl/CPAN/authors/Graham_Barr/

Zu MRTG muß ergänzt werden, daß dieses Tool seine Daten normalerweise durch SNMP-Abfragen bestimmter SNMP-Agenten erhält. Es läßt aber auch zu, statt einer SNMP-Abfrage ein externes Programm (d.h. nicht zum MRTG gehörendes Programm) ausführen zu lassen, welches die ermittelten Daten in einem vorgegebenen Format zurückgibt. Auf Grundlage dessen war es also nur notwendig, ein entsprechendes Programm zu schreiben, daß die benötigten Daten vom jeweiligen Proxycache abfragt und im geforderten Format an MRTG übergibt, so daß selbiger die benötigten HTML-Seiten und Grafiken generieren kann.

1. siehe http://ivs.cs.uni-magdeburg.de/~elkner/webtools/pcms.shtml

Um den Verkehr zum Proxycache und die Anzahl der Abfragen an den Proxycache möglichst gering zu halten und effizient zu gestalten, wurde jedoch ein leicht modifizierter Algorithmus für das PCMS verwendet, da eine Antwort auf eine Anfrage oftmals nicht nur ein Datum sondern gleich mehrere statistische Daten zum Proxycache-Betrieb zurückliefert. Deshalb wurde das Hauptprogramm gps4a.pl wie folgt gegliedert (Steuerfluß):

(1) ermittle alle zur Generierung der HTML-Dateien und Grafiken notwendigen Parameter (mrtg- und echoping-Programmpfad, HTML-Ziel-Verzeichnis, zu testender URL zur Ermittlung der Antwortverzögerungszeiten mittels echoping)

(2) suche einen Eintrag für einen im bisherigen Lauf noch nicht befragten Proxycache aus der Konfigurationsdatei (siehe z.B. Anhang B.3) und ermittle dessen Parameter (Proxycache-Adresse und dessen Kurzbezeichnung, Nutzer-Name, Passwort). Gibt es keinen weiteren Eintrag, dann beende dich.

(3) besteht für den Proxycache eine bestimmte Datei ($cache.lck), gib eine Warnung aus und gehe zu (2), da dieses Programm wahrscheinlich noch immer nicht den vorherigen Lauf beendet hat. Anderenfalls: lege die Datei $cache.lck an.

(4) ermittle die Antwortzeit des Proxycaches durch das Starten des externen Programms echoping

(5) speichere die ermittelten Daten in dem von MRTG geforderten Format in einer separaten Datei ab

(6) ermittle alle anderen gewünschten, statistischen Daten durch Nutzung der in Squid definierten Programmier-Schnittstelle (API)

(7) speichere die ermittelten Daten in je einer separaten Datei in dem von MRTG geforderten Format ab

(8) starte MRTG mit der den Proxycache betreffenden MRTG-Konfigurationsdatei als Parameter (die Konfiguration wurde vor dem erstmaligen Start von gps4a.pl erzeugt, und enthält statt der SNMP-Abfragen den Befehl ‘cat $cache / $data‘, wobei hier $cache eine Variable für den gerade zu betrachtenden Proxycache und $data eine der in (5) bzw. (7) angelegten Dateien).

(9) lösche die Datei $cache.lck

(10) gehe zu (2)

Das Programm gps4a.pl wird für die Proxycaches der MCH per cron Befehl[1] alle 5 Minuten aufgerufen und generiert also im Normallfall alle 5 Minuten die gewünschten HTML-Dateien und Grafiken. Um zu verhindern, daß ein neuer Lauf des Programms

1. siehe ‘man cron‘ auf einem beliebigen UNIX-System

die Daten überschreibt, die in einem noch nicht beendeten vorherigen Lauf des Programms evtl. gerade geschrieben bzw. durch MRTG eingelesen werden, wird eine Synchronisation aller laufenden Programme gps4a.pl mittels einer eindeutigen Datei für jeden Proxycache erreicht (siehe (3) und (9)). Nachteil ist allerdings, daß hierbei zumindest einige statistische Daten unberücksichtigt bleiben, als auch die Gefahr, daß z.B. durch einen Rechnerabsturz die jeweilige Datei nicht mehr rechtzeitig entfernt werden kann, und somit das Programm in nachfolgenden Läufen keine Daten vom jeweiligen Proxycache abfragt. Deshalb wird in solch einer Situation eine Warnung an die Standard-Ausgabe übermittelt, die den zuständigen Nutzer für den Monitor auf den entsprechenden Zustand hinweist und zur Überprüfung der Sachlage auffordert.

Die mittels PCMS generierten WWW-Seiten für ausgewählte Proxycaches der MCH sind unter folgendem URL verfügbar: http://ivs.cs.uni-magdeburg.de/~elkner/stats/proxy/. Stellvertretend dafür sind in Abbildung 3.4 die Statistiken für das LRU Age dargestellt.

Daran ist allerdings auch zu sehen, daß die mittels MRTG erstellten Grafiken nicht für eine detaillierte Analyse ausreichend sein können, da die Genauigkeit der Grafiken bei stark vom Durchschnitt abweichenden Werten deutlich sinkt. Desweiteren muß kritisch angemerkt werden, daß MRTG einen entsprechenden Wert nach einem in der Konfigurationsdatei spezifizierten Intervall ermittelt und davon nur die letzten 600 Werte (bei 5 min Intervall sind das ca. 2 Tage) speichert. Daraus wird die Grafik für den entsprechenden Tag +- n Stunden generiert (1 Pixel bzgl. X-Achse (Zeit) je Intervall). Aus diesen Werten werden wiederum die Durchschnittswerte für die Woche berechnet, wovon wiederum nur die letzten 600 Werte gespeichert werden und je ein Wert einem Pixel bzgl. X-Achse in der Grafik entspricht, d.h. es wird der Durchschnitt während eines 30 min Intervalls je Pixel bzgl. X-Achse abgebildet. Derselbe Algorithmus wird für die Grafik des Monatsstatistiken (1 Pixel = 2 h Durchschnitt) als auch den Jahresstatistiken (1 Pixel = 1 d Durchschnitt) verwendet.

Quintessenz ist, das tatsächlich max. nur die letzten 2 Tage mit höchstmöglicher Auflösung, d.h. der konfigurierten Intervallgröße, aus den erstellten Grafiken bzw. MRTGs Logdatei analysiert werden können. Aus diesem Grund wurde in gps4a.pl eine Option implementiert, mit deren Hilfe die vom Proxycache erhaltenen Werte inkl. Timestamp (Zeit in Sekunden seit 1. Januar 1970 GMT) ebenfalls in je eine separate Da-

tei gespeichert werden. Programme zu deren Analyse sind jedoch vom jeweiligen Nutzer entsprechend seiner Bedürfnisse selbst zu implementieren.

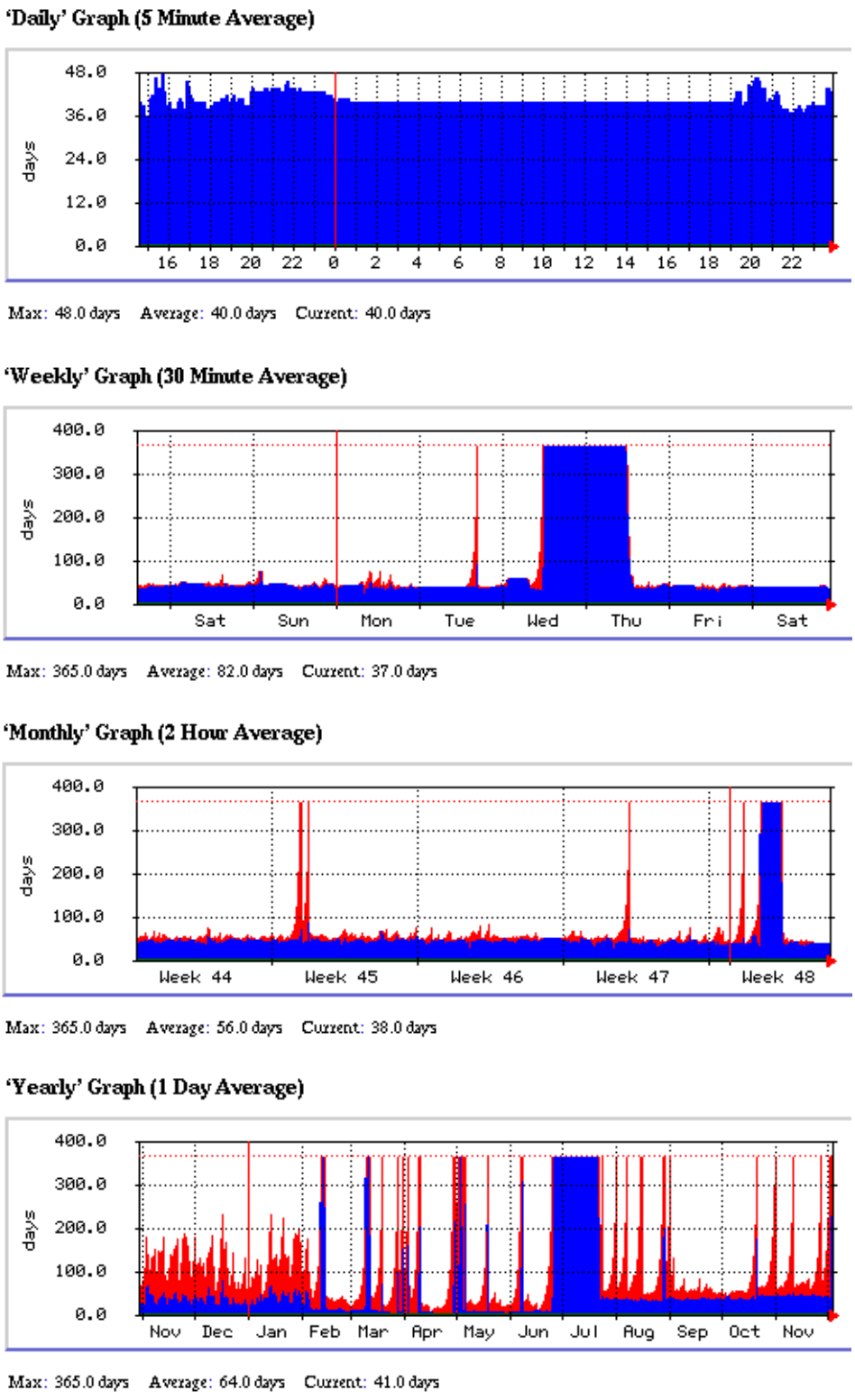

Abbildung 3.4: PCMS / MRTG Statistiken: LRU Age

Ebenfalls ist zu bemängeln, daß weder MRTG noch gps4a.pl sonderlich effizient bzgl. Performance und Nutzung von Systemresourcen sind. So müssen alle Programme inkl. des Perl-Interpreters des PCMS alle 5 Minuten vom System von der Festplatte geladen und abgearbeitet werden. Sowohl gps4a.pl als auch MRTG rufen externe Programme auf, was jeweils einen fork und exec-Systemaufruf und damit einen erhöhten Systemresourcenbedarf (z.B. RAM und CPU-Zyklen) verursacht. Ebenso schreiben beide Programme die aktuell erhaltenen bzw. errechneten Daten in mindestens eine Datei bzw. lesen diese beim Programmstart wieder ein. Das belastet wiederum das System in Hinsicht auf I/O-Durchsatz und die entsprechenden, relativ teuren Systemaufrufe. Des weiteren ist an MRTG zu kritisieren, daß bei jedem Lauf alle Daten für die jeweilige tägliche, wöchentliche, monatliche und jährliche Statistik durch ein externes, sehr rechenintensives Programm (rateup) aus einer Datei neu eingelesen, alle Durchschnittswerte neu berechnet und wieder in der Datei abgespeichert werden.

In der Summe können all die genannten Mängel dazu führen, das u.U. ein oder mehrere Läufe des PCMS noch nicht beendet sind, bevor bereits der nächste Lauf beginnt und damit (wie oben bereits beschrieben) zu Datenverlusten führt. Selbiges konnte bereits mehrmals auf der für die Statistiken der Proxycaches der MCH verantwortlichen Maschine (Sun SPARCstation 20, 2 CPUs, a 60 MHz) festgestellt werden. Ursache war i.d.R., daß kurzzeitig die Verbindung zum Proxycache unterbrochen war, durch Abwarten bestimmter timeouts die Abarbeitung des Programms stark verzögert wurde und da das Programm alle 5 min per cron job gestartet wurde, dieses mehrmals lief und auch mehrmals MRTG inkl. rateup aufgerufen wurde. Obwohl kurze Zeit später die Verbindung zum Proxycache wieder bestand, reichte die Performance der Maschine nicht aus, um die bereits laufenden MRTG-Programme zu befriedigen und die Beendigung derer vor dem Start des nächsten Laufes herbeizuführen. Als Ergebnis dieser Analyse wurde gps4a.pl um den bereits beschriebenen File-Lock-Algorithmus erweitert, der das Starten von MRTG verhindert, sobald ein entsprechendes Lock-File gefunden wird. Inzwischen hat auch der Entwickler von MRTG die Notwendigkeit eines solchen Lock-File-Algorithmus eingesehen und diesen implementiert.

Trotzdessen sind die Programme nach wie vor ineffizient und erzeugen eine relativ hohe Systemlast. Ausgehend von den gesammelten Erfahrungen und neugesammelten Erkenntnissen würde ich für ein neues PCMS einen ...

... Sorry, this page is not available in the public version of this paper ...

... Sorry, this page is not available in the public version of this paper ...

... Sorry, this page is not available in the public version of this paper ...

3.4.3 Empfehlungen für zu beobachtende Proxycache-Status-Daten

Squid liefert über die bereits erwähnte Schnittstelle eine Vielzahl von Informationen (siehe SCM http://ivs.cs.uni-magdeburg.de/cgi-bin/squid.cgi), doch ist ein ständiges

Monitoring aller Informationen (ca. 500 + ca. 10.000 für Filedescriptoren, Netdb, IP-Cache(konfigurationsabhängig))oftmals nicht notwendig bzw. sinnvoll.

Deshalb wurde das Monitoring der Proxycaches der MCH von Anfang an auf die in Tabelle 3.4 aufgelisteten Daten beschränkt. Doch auch hier muß gesagt werden, daß einige dieser Daten nur informativen Charakter haben, und nicht unbedingt notwendig sind, um Aussagen über auftretende oder aufgetretene Störungen bzw. eventuelle Engpässe beim Betrieb des Proxycaches treffen zu können.

Kurzbezeichnung	**Bemerkung**	**Berechnung**
Cache response time	durchschnittliche Antwortzeit des Proxycaches in Millisekunden	absoluter Wert
Filedescriptor usage	Anzahl der Filedescriptors, die zum Zeitpunkt der Abfrage vom Proxycache benutzt werden	absoluter Wert
Unlink requests	durchschnittliche Anzahl von Unlink requests je Minute (Anfragen an den unlink daemon, der für das Löschen von gespeicherten Cache-Objekten zuständig ist)	$(N_{tn} - N_{tn-1}) / 5$ min
DNS server #1 ... #5	Durchschnittliche Anzahl der DNS-Anfragen des Proxycaches je Minute an den externen DNS-Server Nr. #N	$(N_{tn} - N_{tn-1}) / 5$ min
LRU Age	Zeit in Tagen bis zum nächsten Start der LRU expiration (Markierung zwischengespeicherter Objekte als abgelaufen bzw. Löschung selbiger nach dem LRU-Algorithmus)	absoluter Wert
Memory usage	Durch den Proxycache insgesamt genutzter Speicher in Byte.	absoluter Wert
Memory page faults	Durchschnittliche Anzahl von Seiten-Fehlern je Minute.	$(N_{tn} - N_{tn-1}) / 5$ min
Maximum resident size	Maximale residente Größe des Proxycaches in Byte.	absoluter Wert
Select loop	durchschnittliche, zwischen 2 aufeinanderfolgenden Select-Schleifen-Aufrufen vergehende Zeit in Millisekunden (in einem Select-Schleifen-Durchlauf werden alle offenen Filedescriptors abgefragt, und wenn möglich neue Daten gelesen o. geschrieben).	absoluter Wert
Connections TCP, UDP	durchschnittliche Anzahl der TCP - und UDP-Verbindungen je Minute	$(N_{tn} - N_{tn-1}) / 5$ min
TCP: Transfer Total, HTTP	Durchschnittliche Transferrate via TCP insgesamt (d.h. via HTTP, FTP, Gopher) und via TCP und HTTP in Byte je Sekunde	$(N_{tn} - N_{tn-1}) / 5$ min

Tabelle 3.4: Mittels PCMS gesammelte Daten je Proxycache der MCH

Kurzbezeichnung	Bemerkung	Berechnung
TCP: Requests Total, HTTP	Durchschnittliche Anzahl von Anfragen an den Proxycache je Minute via TCP insgesamt (d.h. via HTTP, FTP, Gopher) und via TCP und HTTP	$(N_{tn} - N_{tn-1}) / 5$ min
TCP: Transfer FTP, Gopher	Durchschnittliche Transferrate via TCP und FTP und via TCP und Gopher in Byte je Sekunde	$(N_{tn} - N_{tn-1}) / 5$ min
TCP: Requests FTP, Gopher	Durchschnittliche Anzahl von Anfragen an den Proxycache je Minute via TCP und FTP und via TCP und Gopher	$(N_{tn} - N_{tn-1}) / 5$ min
UDP: Transfer Total, HTTP	Durchschnittliche Transferrate via UDP insgesamt (d.h. via HTTP, FTP, Gopher) und UDP und HTTP in Byte je Sekunde	$(N_{tn} - N_{tn-1}) / 5$ min
UDP: Requests Total, HTTP	Durchschnittliche Anzahl von Anfragen an den Proxycache via UDP insgesamt (d.h. via HTTP, FTP, Gopher) und via TCP und HTTP in Byte je Sekunde	$(N_{tn} - N_{tn-1}) / 5$ min
UDP: Transfer FTP, Gopher	Durchschnittliche Transferrate via UDP und FTP und via UDP und Gopher in Byte je Sekunde	$(N_{tn} - N_{tn-1}) / 5$ min
UDP: Requests FTP, Gopher	Durchschnittliche Anzahl von Anfragen an den Proxycache je Minute via UDP und FTP und via UDP und Gopher	$(N_{tn} - N_{tn-1}) / 5$ min

Tabelle 3.4: Mittels PCMS gesammelte Daten je Proxycache der MCH

Die in Tabelle 3.4 aufgelisteten Meßwerte sind der Einfachheit halber subjektiv nach ihrer Wertigkeit aufgeführt. So wird z.B. die cache response time als das höchste Kriterium für einen nahezu optimal funktionierenden Proxycache angesehen. Wird die gemessene Antwortzeit sehr hoch, d.h. größer 100 ms, sollte möglichst schnell die Ursache für diese großen Verzögerungen gesucht werden (als normal sollten in einem LAN Antwortzeiten kleiner 50 ms gelten). Solche Ursachen könnten z.B. folgende sein:

- nicht ausreichend viele, zur Verfügung stehende Filedescriptors (siehe Statistiken filedescriptor usage)
- der Proxycache ist sehr stark mit dem Löschen von Objekten beschäftigt ist, um Platz für ein oder mehrere neue, relativ große Objekte zu machen (siehe Statistiken unlink requests)
- das LRU Age ist sehr klein und der Proxycache ist hauptsächlich mit der Abarbeitung der LRU expiration beschäftigt (siehe Statistik LRU age)
- zu hoher Speicherbedarf (mehr Speicher allokiert, als RAM auf der entsprechenden Maschine zur Verfügung steht und damit ständiges Swappen)
- es laufen zu wenige Proxycache-DNS-Server und somit können IP-Adressen nicht hinreichend schnell aufgelöst werden (deshalb ist es wichtig, nicht den ersten sondern den zuletzt gestarteten und den via Squid-API erhaltenen letz-

ten Wert für den DNS-Server besonders zu beobachten)

Aus dem Lesen der über die Mailing-Liste versendeten e-mails läßt sich schließen, daß die oben aufgeführten Ursachen in der Tat die häufigsten sind, die zu Performance-Verlusten von Proxycaches führen. Deshalb sollte deren Statistiken die meiste Aufmerksamkeit seitens der Proxycache-Administratoren gewidmet werden.

4 Lastenausgleich (load balancing)

Leider kann grundsätzlich nicht davon ausgegangen werden, daß die im vorherigen Kapitel vorgeschlagenen Maßnahmen ausreichen, um einen hinreichend schnellen Proxycache-Dienst zu garantieren. Das ist z.B. der Fall, wenn ein Proxycache aufgrund sehr vieler Anfragen ständig einen Mangel an freien Filedescriptors, Hauptspeicher oder zu geringe Festplattenkapazität hat oder dessen Netzwerkanbindung nicht den Erfordernissen bzgl. Bandbreite entspricht. Folglich kann hier i.d.R. nur noch eine verteilte Diensterbringung (d.h. die Last wird auf Proxycaches auf mehreren Maschinen verteilt) zu einer Leistungssteigerung bzw. Resourcenentlastung eines oder mehrerer Proxycaches führen.

Im folgenden sollen Möglichkeiten für eine solche Lastenverteilung auf mehrere Proxycaches und am Beispiel der MCH und der dort gesammelten Erfahrungen aufgezeigt werden. Dabei wird speziell mit der Proxy-Konfiguration die Client-Seite, mit der Cache-Hierarchie die Server-Seite (Proxycaches) und mit der IP Network Address Translation das Routing zwischen Client und Server berücksichtigt.

4.1 Proxy-Konfiguration von Clients

Inzwischen gibt es bei fast allen bekannten WWW-Browsern (WWW-Clients) die Möglichkeit, die Nutzung zumindest eines Proxies für jedes der vom Client unterstützten Protokolle (i.d.R. HTTP, FTP, Gopher, Wais und SOCKS) zu konfigurieren. Dabei wird oftmals eine manuelle oder / und eine automatische Konfiguration des Browsers unterstützt.

4.1.1 Die manuelle Proxy-Konfiguration

Bei der manuellen Proxy-Konfiguration eines WWW-Browsers kann i.d.R. für jedes unterstützte Protokoll die IP-Adresse und der zu nutzende Port **eines** Proxies eingetragen werden. Ebenso können oftmals auch Domains angegeben werden, von denen die angeforderten Objekte direkt heruntergeladen werden, der Proxy also nicht in Anspruch genommen werden soll. Abbildung 4.1 zeigt beispielsweise die Dialog-Fenster

für die manuelle Proxy-Konfiguration[1] der am häufigsten genutzten WWW-Browser Netscape Navigator/Communicator (i.w. kurz als Netscape bezeichnet) und des Microsoft Internet Explorers (MSIE).

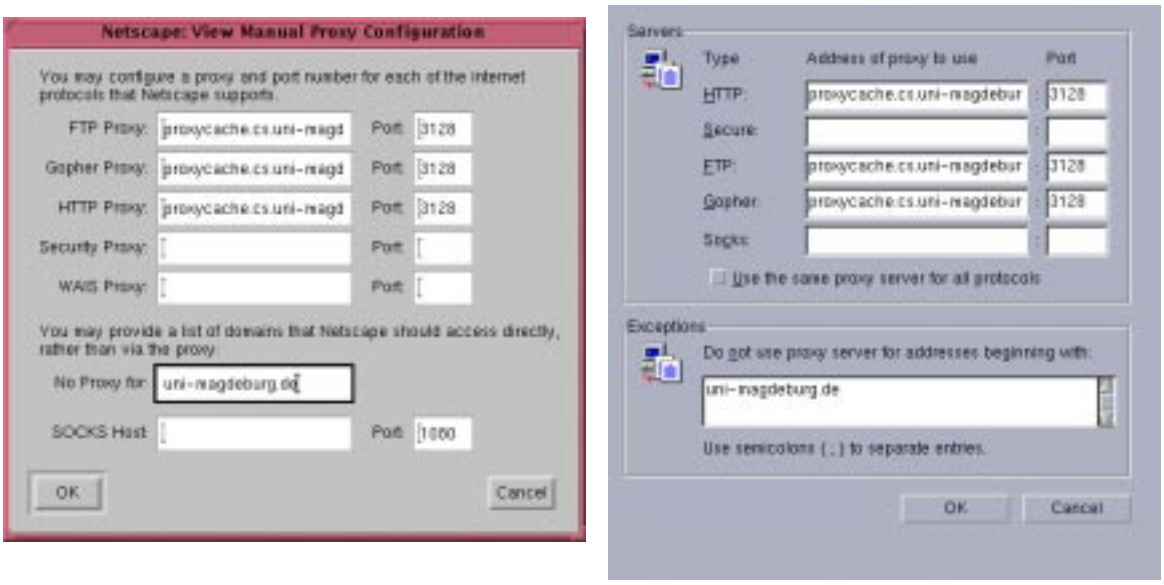

Abbildung 4.1: Manuelle Proxy-Konfiguration von Netscape (links) und MSIE (rechts)

Damit kann diese Möglichkeit genutzt werden, eine Lastenverteilung auf verschiedene Proxycaches zu erreichen, indem die Nutzer instruiert werden, jeweils verschiedene Proxycaches einzutragen.

Das Problem bei der praktischen Umsetzung ist jedoch, wie diese Verteilung erfolgen sollte. Bei der MCH wurde dazu die Summe aller Anfragen an alle Proxycaches ermittelt. Anschließend wurde unter Berücksichtigung der vorhandenen Netzwerk-Topologie der Universität Magdeburg eine Zuteilung der Proxycaches bzgl. der Zugehörigkeit der Maschinen zu den einzelnen Instituten, Fakultäten oder Einrichtungen vorgenommen. Da den Instituten, Fakultäten und sonstigen Einrichtungen der Universität Magdeburg meist vollständige IP-Subnetze zugeordnet sind (z.B. Fakultät für Informatik 141.44.18.0/24 ... 141.44.30.0/24, Fakultät für Mathematik 141.44.70.0/24 ... 141.44.75.0/24) wurde ebenfalls die Zuteilung der Proxycaches bzgl. der IP-Subnetze an der Universität Magdeburg festgelegt (Tabelle 4.1 zeigt die aktuelle Zuteilung

1. Detailliertere Informationen zur Proxy-Konfiguration verschiedener WWW-Browser können u.a. unter folgendem URL nachgelesen werden: http://ivs.cs.uni-magdeburg.de/~elkner/proxy/Knowledge/config.shtml

der Proxycaches) und dies solange verfeinert, bis eine verhältnismäßig gleichmäßige Auslastung der Proxycaches bzgl. Anfragen je Stunde erreicht wurde. Die Informationen wurden in universitätsinternen Newsgroups (unimd.announce, unimd.cs.announce) bekanntgegeben, mit der Bitte an die Nutzer, den für ihre Maschine bestimmten Proxycache zu nutzen und explizit die Nutzung des Proxycaches für alle URLs, die sich auf die Domain uni-magdeburg.de beziehen abzuschalten (siehe Abbildung 4.1). Ebenfalls wurden entsprechende HTML-Seiten erstellt[1] und Verweise darauf in verschiedenen HTML-Seiten von WWW-Servern der Universität Magdeburg eingetragen.

IP-Adresse der Maschine, auf der der WWW-Browser genutzt wird bzw. Bereich	**zu verwendender Proxycache**	**Port**
141.44.1.1 ... 141.44.17.254	www-cache.uni-magdeburg.de	3128
141.44.18.1 ... 141.44.69.254	proxycache.cs.uni-magdeburg.de	3128
141.44.70.1 ... 141.44.254.254	www-cache.uni-magdeburg.de	3128
149.203.1.1 ... 149.203.254.254	proxycache4.med.uni-magdeburg.de	3128
Externe Proxycaches (Nachbarn)	proxycache.cs.uni-magdeburg.de	3128
ISG, ITI, IVS, IWS, FIN, IFN	proxycache.cs.uni-magdeburg.de	3128
URZ, Mathematik, Maschinenbau, Wirtschaftswissenschaften, Wohnheime u.a.	www-cache.uni-magdeburg.de	3128
Medizinische Akademie	proxycache4.med.uni-magdeburg.de	3128

Tabelle 4.1: Zuteilung der Proxycaches der Universität Magdeburg

Nachteil der manuellen Proxy-Konfiguration ist allerdings, daß für jedes Protokoll immer nur ein Proxycache eingetragen werden kann. Fällt dieser aus ist der Download von WWW-Objekten nur möglich, wenn die Nutzung des Proxycaches im Browser explizit abgeschalten wird. Ebenso müssen die Nutzer jedesmal informiert und gebeten werden, ihre Einstellungen im WWW-Browser zu ändern, falls es Veränderungen bzgl. der Nutzung von Proxycaches ergibt. Last but not least ist das Verhalten des Browsers bei manueller Konfiguration recht unflexibel bzgl. der Nutzung von Proxycaches, wenn es darum geht, je nach anzufordernder URL einen bestimmten oder keinen Proxycache zu nutzen. Deshalb wird an der Universität Magdeburg die Nutzung der

1. siehe http://ivs.cs.uni-magdeburg.de/~elkner/proxy/Knowledge/config.shtml

automatischen Proxy-Konfiguration favorisiert, die im nächsten Abschnitt näher erläutert wird.

4.1.2 Automatische Proxy-Konfiguration

Die automatische Proxy-Konfiguration ist die einfachste Methode, Clients zu instruieren, bestimmte oder alle Anfragen an einen bestimmten Proxycache oder direkt an den zuständigen Server zu senden. Derzeit wird die automatische Proxy-Konfiguration jedoch nur von Netscape und MSIE unterstützt, was jedoch nicht als sonderlich tragisch angesehen wird, da diese die weltweit am meisten genutzten WWW-Browser sind (schätzungsweise >= 95%), was so auch auf die Universität Magdeburg übertragbar ist.

Prinzip der automatischen Proxy-Konfiguration ist, daß ein sogenanntes *proxy autoconfig file* (siehe auch [45]) eigenständig durch den WWW-Browser von einem anzugebenden WWW-Server heruntergeladen wird, welches den *Content-Type: application/x-ns-proxy-autoconfig* haben und eine in *Javascript* [46] geschriebene Funktion namens *FindProxyForURL(url, host)* enthalten muß. Diese Funktion wird bei jeder URL-Anfrage abgearbeitet und muß als *Ergebnis eine Liste von zu nutzenden Proxies* liefern. Anschließend wird die Anfrage an den in der Liste zu Beginn stehenden Proxy gestellt. Ist dieser nicht erreichbar, wird die Anfrage an den nächsten in der Liste befindlichen Proxy gestellt. Ist keiner der in der Liste befindlichen Proxies erreichbar und enthält die Liste nicht das spezielle Schlüsselwort *DIRECT* (weist den Browser an, das Objekt direkt vom Originalserver zu holen), wird der Nutzer gefragt, ob die derzeitige Proxy-Konfiguration temporär ignoriert und alle Objekte direkt von den Originalservern geholt werden sollen. Ist dies der Fall, fragt Netscape den Nutzer im 20 minütlichen Intervall, ob probiert werden soll, wieder Proxies zu benutzen. War ein oder mehrere Proxies aus der Liste nicht erreichbar, versucht Netscape im Intervall von 30 Minuten, den in der Liste am weitesten vorn stehenden, erreichbaren statt den bisherigen Proxy zu nutzen.

Das aktuelle an der Universität Magdeburg verwendete proxy autoconfig file (PAC) hat folgenden Inhalt und wurde zwecks Lastverteilung auf drei WWW-Servern der Universität installiert:

```
// Automatische Proxy Configuration fuer Clients der Uni Magdeburg
// (C) by Jens Elkner <elkner@ivs.cs.uni-magdeburg.de>
//
// http://www.cs.uni-magdeburg.de/uni-md.pac
// http://ivs.cs.uni-magdeburg.de/uni-md.pac
// http://www.math.uni-magdeburg.de/uni-md.pac
//
// see http://ivs.cs.uni-magdeburg.de/~elkner/proxy/Knowledge/config.shtml
//
// Last Update: 18.07.98
//
function FindProxyForURL(url, host) {
URZcache = „PROXY www-cache.uni-magdeburg.de:3128; „ +
          „PROXY proxycache.cs.uni-magdeburg.de:3128; „ +
            „DIRECT“;
FINcache = „PROXY proxycache.cs.uni-magdeburg.de:3128; „ +
            „PROXY www-cache.uni-magdeburg.de:3128; „ +
            „DIRECT“;
TESTcache = „PROXY proxycache1.cs.uni-magdeburg.de:3128; „ +
             „PROXY proxycache.cs.uni-magdeburg.de:3128; „ +
             „DIRECT“;
          if (isPlainHostName(host) ||
             dnsDomainIs(host, „.uni-magdeburg.de“) ||
             isInNet(host, „141.44.0.0“, „255.255.0.0“) )
          //  || (url.indexOf(„\?“) > -1) )
            return „DIRECT“;
          else {
            clientIP = myIpAddress();
            subNet = clientIP.substring(7,clientIP.lastIndexOf(„.“));
            if ( subNet < 18 || subNet > 75) return URZcache;
            if ( subNet < 70 ) return FINcache;
            if ( url.indexOf(„msnprhtml“) > -1 ) return „DIRECT“;
            else return URZcache;
          }
}
```

Dies bewirkt, daß alle Objekte, die von Servern der Domain uni-magdeburg.de bzw. dem Class B Netz 141.44.0.0/16 (Intranet der Universität Magdeburg) geladen werden sollen, direkt von diesen Servern abgefragt werden. Werden Objekte aus anderen IP-Netzen abgefragt, wird je nach IP-Adresse der Maschine (siehe Tabelle 4.1), auf der der WWW-Browser läuft, die Liste URZcache oder FINcache als Ergebnis zurückgeliefert.

Ein Sonderfall liegt bei den IP-Subnetzen 141.44.70.0/24 ... 141.44.75.0/24 inklusive (Fakultät für Mathematik) vor. Hier wird der Browser instruiert, alle Objekte, die die Zeichenkette „msnprhtml“ in ihren URLs enthalten, nicht von einem in der Liste URZ-cache enthaltenen Proxycaches sondern direkt vom Originalserver abzufragen. Das ist notwendig, da der Zugriff auf die WWW-Seiten der Suchmaschine der American Mathematical Society (AMS, http://ams.mathematik.uni-bielefeld.de/mathscinet/) nur für die oben genannten Subnetze entsprechend dem Lizenzabkommen erlaubt ist. Pro-

blem dabei ist, daß sich kein Proxycache in den Subnetzen der Fakultät für Mathematik befindet und somit bei Nutzung eines Proxycaches grundsätzlich der Zugriff durch den jeweiligen WWW-Server der AMS verweigert wird.

Um die Komplexität der Funktion so gering wie möglich zu halten, wurden die „gesperrten“ URLs analysiert und herausgefunden, daß diese Seiten immer die Zeichenkette „msnprhtml“ enthalten und somit diese als Kriterium für den direkten Zugriff ausgewählt. Alternativ bestünde auch die Möglichkeit, auf die Hostadresse des Servers zu testen. Dies würde jedoch bedeuten, daß eine Liste aller Server, die diese Seiten spiegeln, gewartet werden müßte, was wiederum zusätzlichen Mehraufwand bzgl. Wartung der Liste und höheren Rechenaufwand beim Vergleich der Host-Adresse in der URL mit denen in dieser Liste enthaltenen Host-Adressen bedeutet. Ebenso war zu berücksichtigen, daß viele andere Objekte dieser WWW-Server für die Öffentlichkeit zugänglich sind und damit die Nutzung eines Proxycaches durchaus sinnvoll erscheint.

Derzeit ist die Zeile „| | (url.indexOf("\?") > -1))“ im PAC mittels „/ /“ auskommentiert, was bewirkt, daß die Browser auch argumentbehaftete Anfragen an den Proxycache schicken, obwohl dieser standardmäßig die Ergebnisse solcher Anfragen nicht zwischenspeichert. Hintergrund ist die im vorherigen Kapitel unter „Redirector“ erwähnte Umleitung von Anfragen. Sollte es wider Erwarten zu Überlastungserscheinungen auf einem Proxycache kommen, sollte überlegt werden, ob diese Zeile nicht wieder aktiviert wird und damit die WWW-Browser instruiert werden, alle argumentbehafteten Anfragen direkt an die Originalserver zu stellen und so den oder die Proxycaches wieder etwas zu entlasten.

An dem hier aufgezeigten Fallbeispiel sollte zu erkennen sein, daß die automatische Proxy-Konfiguration sehr gut dazu geeignet ist, eine Lastenverteilung auf verschiedene Proxycaches zu erreichen. Dabei ist diese Lastenverteilung bei weitem nicht nur wie im obigen Beispiel an die IP-Adresse eines Clients gebunden, sondern es sind auch bedeutend komplexere Mechanismen, wie z.B. das in [47] erwähnte URL-Hashing möglich.

Desweiteren ist es sehr einfach, Spezialfälle (wie z.B. AMS-Suchmaschine) zu berücksichtigen und die Nutzung der Proxycaches sehr flexibel und für den Nutzer relativ transparent zu gestalten.

Nachteil der automatischen Proxy-Konfiguration ist, daß sie nicht vollständig automatisch ist, da der Nutzer erst einmal die entsprechende URL für das Script für die automatische Proxy-Konfiguration in seinen Browser eintragen muß, um dann die FindProxyURL Funktion nutzen zu können. Zumindest bei im Netzwerk installierten Browsern (d.h., eine oder mehrere bestimmte Nutzergruppen laden das Programm von der gleichen Platte) wäre z.B. eine zentrale Konfigurationsdatei wünschenswert, die beim Start des Browser automatisch eingelesen wird und damit dem Nutzer das manuelle Eintragen von Proxycaches bzw. dem automatischen Proxy-Konfigurations-URL erspart. Um dies dennoch zumindest für alle UNIX-Workstation-Nutzer im IVS und der FIN-Pools zu erreichen, wurde ein Perl- Script geschrieben, was regelmäßig per cron job gestartet wird und die notwendigen Einträge in der Netscape-Konfigurationsdatei der Nutzer vornimmt[1]. Möglich ist dies, da alle Nutzerdaten auf zentralen File-Servern gehalten werden und der Network Information Service (NIS) zu Lokalisierung selbiger genutzt werden kann. Da auf Windows-95/NT-Rechnern i.d.R. die Browser-Software und deren Konfigurationsdateien dezentral installiert sind, d.h. lokal auf der jeweiligen Maschine, ist auf jeden Fall ein bedeutend höherer Aufwand zu betreiben, um ein Verfahren auszuarbeiten, das ähnliche Funktionalität wie das oben beschriebene Verfahren für UNIX-Workstations bietet.

4.2 IP Network Address Translation (NAT)

Ist es nicht oder nur sehr schlecht möglich, alle Nutzer zu informieren bzw. zu bewegen, ihre Browser so einzustellen, daß die vorgegebenen Proxies genutzt werden, bietet die IP Network Address Translation (NAT) [48] eine Möglichkeit, auch ohne spezielle Browsereinstellungen durch den Nutzer Proxies nutzen zu lassen. Deshalb wird NAT im Zusammenhang mit Proxies bzw. Proxycaches oft auch als *Transparent Caching/Proxying* bezeichnet. Prinzip dabei ist, daß alle IP-Pakete, die an einen gewissen Port (normalerweise Port 80 für HTTP) einer Maschine geschickt werden, zu einem bestimmten Proxy umgeleitet werden und dieser die Pakete entsprechend seiner Funktionalität behandelt. Damit Squid die umgeleiteten Pakete richtig interpretiert, müssen alle Versionen <= 1.2b25 gepatcht werden. Alle höheren Versionen von Squid

1. Das Script ist auf der beigefügten Diskette unter dem Namen proxy-eintrag.pl enthalten.

unterstützen generell transparent Caching / Proxying, wenn die Software mit dem Schalter *--enable-ipf-transparent* konfiguriert und anschließend kompiliert wird.

Die Umleitung (Redirection) der Pakete zum Proxy kann generell nur durch IP-Router erfolgen, die sich zwischen Quelle und Ziel der eigentlichen Anfrage befinden. Router in diesem Sinne können also Workstations oder PCs mit geeignetem Betriebssystem und zusätzlicher Software oder NAT-fähige Netzwerk-Komponenten wie Layer-4-Switches und Router sein.

Als geeignete IP-Filtersoftware wird i.a. für Solaris als auch Free-, Open-, NetBSD, IRIX und Linux *IP Filter* (siehe http://cheops.anu.edu.au/~avalon/ip-filter.html) bzw. für Linux ebenfalls das *Linux IP Firewall and Accounting - ipfwadm* (siehe http://www.xos.nl/linux/ipfwadm/) angesehen. Die genannten Betriebssysteme unterstützen auch ohne diese Software bereits das Routing von IP-Paketen.

Als NAT-fähige Router sind derzeit nur CISCO Router mit dem Betriebssystem IOS 11.2 inkl. Plus-Software-Paket bekannt. Alle IOS Versionen >= 12.x sollen auch ohne Plus-Software-Paket NAT unterstützen (siehe http://www.cisco.com/univ-src/cc-den/data/doc/software/11_2/cnp1/5cip.htm#REF30065). Layer-4-Switches gibt es inzwischen bei jedem größeren Netzwerkkomponentenhersteller, wobei in diversen Proxycacheadministrator-Mailinglisten öfters Alteon Networks ACEdirector 1 und 2 positiv erwähnt wurden.

Auf die Art und Weise der Konfiguration der entsprechenden Software bzw. Netzwerkkomponenten soll hier nicht eingegangen werden, da dies je nach Hersteller oftmals unterschiedlich ist und damit den Rahmen dieser Arbeit sprengen würde[1]. Fakt ist, das durch NAT nicht nur ein transparent Caching / Proxying sondern auch eine Client-IP-basierte Lastverteilung von Anfragen auf verschiedene Proxycaches möglich ist. Nachteil dabei ist, daß jetzt der Nutzer keinen Einfluß mehr darauf hat, ob ein oder mehrere Proxycaches und welcher Proxycache genau genutzt werden soll, was besonders tragisch bei einem Fehlverhalten eines Proxycaches ist. Ebenso nachteilig ist, daß bei diesem Verfahren FTP-Pakete nicht an Squid umgeleitet werden können, da der Browser beim Einsatz von NAT nichts von einem Proxy weiß (Transparenz) und deshalb nicht das HTTP-Protokoll sondern direkt das FTP-Protokoll verwendet, um das

1. siehe u.a. http://squid.nlanr.net/Squid/FAQ/FAQ-17.html

entsprechende Objekt herunterzuladen. Da Squid aber ein reiner HTTP-Proxy ist, kann er FTP-Paket-Anfragen nicht ordnungsgemäß verarbeiten. Analog verhält es sich mit Gopher- und Wais-Anfragen. Somit ist klar, daß Squid nur noch für via HTTP angeforderte Objekte verwendet wird, was die Hitrate bzgl. Volumen sehr stark reduzieren würde. Des weiteren muß bei NAT darauf geachtet werden, daß die verwendete NAT-Software die Möglichkeit bietet, Pakete zum Originalserver weiterzuleiten, wenn der definierte Proxycache nicht erreichbar ist oder es muß ein eigenständiges Programm geschrieben werden, um dies zu gewährleisten.

Als letzter Nachteil soll hier das relativ feste Schema der Umleitung genannt sein. NAT erlaubt es nur anhand der Zieladresse bzw. der Quelladresse und / oder -Port von Paketen, diese zu einem bestimmten Proxy umzuleiten. D.h. evtl. besser skalierbare Mechanismen wie z.B. URL-Hashing bei der automatischen Proxy-Konfiguration sind hier nicht möglich.

Aufgrund dieser Nachteile und der Überlegung, daß die Nutzung von Proxycaches derzeit den Nutzern der Universität nur als „WWW-Beschleuniger“ empfohlen, nicht jedoch vorgeschrieben wird, wurde bisher auf den Einsatz der NAT verzichtet. Wird in Zukunft die zur Verfügung stehende Bandbreite zwischen dem Intranet der Universität Magdeburg und dem B-WiN des DFN (Extranet) ein Problem, muß sicher überlegt werden, ob durch den Einsatz von NAT das Transportvolumen zwischen Intra- und Extranet reduziert werden kann. In diesem Fall wird eine Lastenverteilung unumgänglich sein, da dann bedeutend mehr Anfragen verarbeitet werden müssen (lt. subjektiver Schätzung nutzen derzeit nur ca. 10-20% aller Nutzer an der Universität Magdeburg Proxycaches).

Da NAT relativ große Nachteile gegenüber einer manuellen Proxy-Konfiguration hat und die automatische Proxy-Konfiguration weitaus flexibler als die manuelle ist, sollte auch beim Einsatz von NAT weiterhin die Verwendung der automatischen Proxy-Konfiguration durch alle Nutzer primäres Ziel sein, um eine bestmögliche Lastenverteilung und damit Ausnutzung der Vorteile des Proxycaching zu erreichen.

4.3 Cache-Hierarchie

Das Problem der Lastverteilung tritt natürlich auch in Proxycache-Hierarchien auf. Hier eine möglichst optimale Lösung bzgl. einer möglichst gleichmäßigen Lastverteil-

lung im Verhältnis zu den zur Verfügung stehenden Resourcen wie z.B. dem vorhandenen Plattenplatz und Hauptspeicher, der CPU-Leistung und Bandbreite zu finden, ist relativ schwierig und deshalb schon seit mehreren Jahren Gegenstand unterschiedlichster Untersuchungen, Ausgangspunkt von Diskussionen in Workshops und wissenschaftlichen Abhandlungen. Das hier weiterhin Forschungsbedarf besteht, wird durch den Fakt widergespiegelt, daß es derzeit nur zwei etwas weiter verbreitete, in Produktionssystemen eingesetzte Protokolle zum load balancing gibt, die keinen experimentellen Status besitzen: das WCCP und CARP (siehe Abschnitt „Das Web Cache Control Protocol (WCCP)“ auf Seite 36 ff. und „Das Cache Array Routing Protocol (CARP)“ auf Seite 38 ff.). Das WCCP wird auf CISCO Cache Engines, das CARP auf Microsoft-Proxycaches eingesetzt.

Ab Squid Version >= 1.2b25 existiert auch eine experimentelle Implementation des CARP, die durch das Setzen des Schalters *--enable-carp* beim Konfigurieren und dem anschließenden Kompilieren der Software aktiviert werden kann.

Leider gab es bisher keine zuverlässigen Aussagen über die Qualität des Lastenausgleichs, der durch die Verwendung der oben genannten Protokolle erreicht wird. Da dies jedoch als wichtige Eigenschaft eines Proxycache-Systems erachtet wird, konnte nicht darauf verzichtet werden, zumindest ansatzweise Tests durchzuführen, die diesbezüglich eine zumindest grobe Aussage zulassen.

Um also zu überprüfen, wie gut ein load balancing bzgl. gestellter Anfragen als auch bzgl. übertragener Bytes mittels eines bestimmten Algorithmus ausfallen könnte, wurde im Rahmen dieser Arbeit die Hauptroutine von Squid umprogrammiert und durch geeignete, neu programmierte Module ergänzt und somit zum Testprogramm namens *lb-checker* für verschiedene Lastenausgleichs-Algorithmen umfunktioniert (ca. 1000 Zeilen Quellcode). Da eine Evaluierung im Online-Betrieb zu aufwendig wäre und keine Langzeit-Aussagen zuließe, wurden als Datenquellen die „access_log“-Dateien[1] [49] der Monate Januar bis November 1998 verwendet, denn wird davon ausgegangen, daß sich das Browserverhalten der Nutzer in den nächsten Monaten nicht sonderlich ändert, so müßte ein Algorithmus, der für den bisherigen Zeitraum für einen guten Lastenausgleich gesorgt hat, auch in den nächsten Monaten

1. Dateien, in denen z.B. jede Anfrage an den Cache inkl. IP-Adresse des Clients, URL, Anfrage-Methode, übertragene Anzahl von Bytes und Zeitpunkt der Anfrage enthalten ist

für selbigen sorgen. Somit wurde die in dem genannten Zeitraum durch WWW-Clients erzeugte Last nachgestellt (ca. 0.8 ... 1.4 mio Anfragen / Monat und Transfervolumen von ca. 8... 14 GB / Monat) und analysiert. Dabei wurden die Anzahl der Anfragen und transferierten Bytes je Cache von Monatsbeginn bis Monatsende kumuliert und im Intervall von jeweils 2 Stunden abgefragt und in einer Datei protokolliert. Um eine Verfälschung der Ergebnisse zu vermeiden, wurden nur die Anfragen berücksichtigt, die nicht von Nachbar-Proxycaches oder Children kamen und nicht cache_object als Protokoll spezifiziert hatten. Ebenso wurden alle Fragen vom Rechner, auf dem das PCMS läuft ausgeschlossen. Die Details und Ergebnisse zu den durchgeführten Tests werden in den folgenden Abschnitten kurz dargestellt und sollen als Anregungen für weitere Untersuchen dienen.

4.3.1 CARP und Squid-CARP

In Test 1 wurde das CARP-Protokoll, in Test 2 das leicht modifizierte CARP-Protokoll von Squid Version >= 1.1b25 (hier sind alle bitweisen Rotationen nach links um N Bit durch ein bitweise Verschieben nach links um N Bit ersetzt worden) getestet. Die verwendeten Parameter sind in Tabelle 4.2 dargestellt. Entsprechend dem verwendeten load factor ist zu erwarten, daß alle Proxycaches gleichmäßig beansprucht werden müßten, d.h ca. 25% aller bis zum Zeitpunkt der Messung gestellten Anfragen bzw. übertragenen Bytes müßten auf jeden Proxycache entfallen sein. Wie die Ergebnisse

Proxycache Adresse	**Load factor**
leipzig.www-cache.dfn.de	0.25
nuernberg.www-cache.dfn.de	0.25
berlin.www-cache.dfn.de	0.25
hamburg.www-cache.dfn.de	0.25

Tabelle 4.2: Parameter für load balancing Test: CARP

zeigen (siehe Abbildung A.4 ... Abbildung A.23 in Anhang A.6.1) , ist das sowohl bei CARP als auch Squid-CARP leider nicht der Fall. Erst nach ca. 1-3 Tagen wird eine annähernd gleichmäßige Auslastung der Proxycaches erkennbar (Auslastung zwischen 20-30%). Das lies vermuten, daß CARP innerhalb eines relativ kleinen Meßintervalls wahrscheinlich nur sehr schlecht eine gleichmäßige Auslastung aller Caches bewirkt.

Um dies zu verifizieren, wurde jedes Meßintervall aus Test 1 und Test 2 separat betrachtet und in Abbildung A.24 ... Abbildung A.43 dargestellt. Hier ist eine relativ starke Streuung um die 25% Marke (besonders bei der übertragenen Anzahl von Bytes) zu erkennen, was die vorherige Vermutung bestätigt und folgern läßt, daß über kürzere Zeiträume hinweg mittels CARP eine Überlastung von Caches nicht ausgeschlossen und nur ansatzweise ein load balancing erreicht werden kann. Ähnliche Ergebnisse[1] liefern auch die analog durchgeführten Tests 3 und 4, bei denen die access log Dateien von www-cache.uni-magdeburg.de benutzt wurden.

Desweiteren kann festgestellt werden, daß bzgl. Anfragen mit Squid-CARP eine etwas gleichmäßigere Auslastung als mit CARP erreicht werden würde, sich aber mit Squid-CARP eine annähernd gleichmäßige Auslastung i.d.R. etwas später einstellt, als mit CARP.

4.3.2 Simple Hashes

Ausgehend von CARP und dem im Super Proxy Script [47] dargestellten Algorithmus wurde vom Autoren ansatzweise untersucht, inwieweit andere, daraus abgeleitete Algorithmen für einen Lastenausgleich tauglich sind. Ähnlich wie CARP wurde hier per Squid-Konfigurationsdatei den jeweiligen Caches ein Lastfaktor zwischen 0 und 1 zugeteilt, mit der Bedingung, daß die Summe aller Lastfaktoren = 1 ist. Jeder Lastfaktor wurden anschließend auf den Bereich 0 .. 100 abgebildet, in dem er mit 100 multipliziert und anschließend die Summe aller bisher so ermittelten Produkte aufaddiert wurde (siehe Tabelle 4.3).

Proxycache	**Lastfaktor aus squid.conf**	**im Hashing benutzter Wert**
nuernberg.www-cache.dfn.de	0.25	25
berlin.www-cache.dfn.de	0.25	50
hamburg.www-cache.dfn.de	0.25	75
leipzig.www-cache.dfn.de	0.25	100

Tabelle 4.3: Simple Hashing: Initialisierung der Vergleichswerte

In jedem untersuchten Algorithmus wird für einen oder mehrere Teile des URL eine Summe errechnet, die sich durch Addition der ASCII-Werte aller Zeichen in der jewei-

1. siehe beiliegender CD unter /results/carp/test2 bzw. /results/carp/test2

ligen Teilzeichenkette ergibt. Diese Summe wurde mittels Operation *modulo 100* auf einen Hash-Wert zwischen 0 .. 100 abgebildet und mit den Lastfaktoren der Proxycaches (beim kleinsten beginnend, beim höchsten endend) verglichen. Als Ergebnis wird der Proxycache ausgewählt, für den als erstes die Aussage „Hash-Wert ist kleiner oder gleich dem eigenen Lastfaktor“ wahr ist. Beispielsweise verdeutlicht folgender Quellcode diesen Algorithmus anhand der fileSelectPeer Funktion, indem der Hash-Wert aus dem in dem URL befindlichen Dateinamen gebildet wird (alle Zeichen hinter dem zuletzt auftretenden / (Slash)) und anschließend mit dem Lastfaktoren der einzelnen Proxycaches verglichen wird. Als Ergebnis wird ein Zeiger auf den für den entsprechenden Proxycache angelegten Datenbereich[1] zurückgegeben.

```
peer *
fileSelectPeer(request_t * request)
{
    String s;
    char *start, *end, *go;
    peer *tp;
    unsigned long url_hash = 0;

    s = request->urlpath;
    start  = strrchr(s.buf,'/');
    if (! start )
        start = s.buf ;
    end = start + s.len;
    for (go=start; go < end; go++)
        url_hash += *go;
    url_hash %= 100;
    for (tp = Config.peers; tp; tp = tp->next) {
        if (url_hash <= tp->carp.load_factor ) {
            debug(1001, 3) ("fileSelectPeer: selected host %s with score %l d\n",
                                        tp->host, url_hash);
            return tp;
        }
    }
}
```

Folgende Abwandlungen dieses Algorithmus wurden benutzt:

Test	**Hash-Wert aus**	**Beispiel** http://homes.cls.net/~Bjoern.Deutschmann/news2.html
5	host-part	homes.cls.net
6	directory	/~Bjoern.Deutschmann/

Tabelle 4.4: Simple Hash Algorithmen zum Lastenausgleich

1. siehe beigefügte CD-ROM: /squid-2.1.PATCH2/src/structs.h

Test	**Hash-Wert aus**	**Beispiel** http://homes.cls.net/~Bjoern.Deutschmann/news2.html
7	file	news2.html
8	host-part, directory	homes.cls.net + /~Bjoern.Deutschmann/
9	host-part, file	homes.cls.net + news2.html
10	directory, file	/~Bjoern.Deutschmann/news2.html

Tabelle 4.4: Simple Hash Algorithmen zum Lastenausgleich

Eine grafische Auswertung analog zu Test 1 und 2 ergibt, daß auch diese Algorithmen nicht zum gewünschten Erfolg führen. Es wird zwar auch hier langzeitmäßig eine Verteilung der Last auf die Proxycaches im Bereich +- 5% der vorgegebenen Marken (25%) erreicht (wobei i.d.R. etwas schlechter als CARP bzw. Squid-CARP), aber auch im kurzfristigen Bereich sind sehr starke Abweichungen davon zu erkennen.

Die Grafiken und Daten zu allen durchgeführten Tests sind auf der beigefügten CD-ROM enthalten (/results/carp) und können vom interessierten Leser zur weiteren Auswertung studiert bzw. verwendet werden. Auf die Abbildung der zugehörigen Grafiken wurde deshalb an dieser Stelle verzichtet.

4.3.3 Vorschläge für weitere Tests

Wie die durchgeführten Tests bereits andeuten, ist derzeit noch kein optimaler Algorithmus gefunden worden, der redundantes Cachen von Daten vermeidet, die gewünschten Proxies lt. Vorgabe auslastet und hinreichend schnell ist, um bedeutende Verzögerungen zu verhindern. Somit wird es weiterhin eine Aufgabe der Wissenschaft und Technik sein, hier nach besseren, effizienten Lastausgleichsverfahren zu suchen, um weiter die Qualität des WWW-Dienstes zu verbessern als auch die Kosten dafür niedrig zu halten bzw. zu senken.

Vorstellbar in Hinsicht auf weitere Untersuchungen wäre zum Beispiel die Analyse bisheriger access log files bzgl. des domain names im host-part des URL, oder die Dateierweiterung (file extension) der im URL angegebenen Datei, dem verwendeten Protokoll und deren Korrelation zu den Objektgrößen, Anzahl der HITs, und MISSes, Anzahler der Anfragen allgemein und Mustererkennung (pattern matching) in URLs. Um generell einen Überblick bzgl. der genannten statistischen Werte zu haben, wurde vom Autoren bereits ein Programm namens *jesalfa*[1] entwickelt, welches access log fi-

1. siehe http://ivs.cs.uni-magdeburg.de/~elkner/webtools/jesalfa/

les analysiert und die Ergebnisse je nach Bedarf im HTML-Format oder / und Textformat ausgibt. Ein weiteres Programm könnte z.B. diese Ergebnisse weiter auswerten und zur Entwicklung neuer Hash-Funktionen zum load balancing verwenden. So zeigen die mittels jesalfa gewonnen Ergebnisse (siehe beigefügte CD-ROM /results/jesalfa/cache[0]/clients/), daß es z.B. die Anzahl der Zugriffe bzw. transferierten Bytes auf bestimmte Domains über Monate hinweg einer bestimmten Regelmäßigkeit unterliegen und somit eine Grundlage für neue Lastausgleichsverfahren bzw. effizientes Caching relevanter Objekte sein könnten. Ebenso ist zu erkennen, daß auch bzgl. Dateierweiterungen eine gewisse Regelmäßigkeit existiert und damit eine Basis für weitere Forschungen vorhanden ist.

Die bis zum Dezember 1998 gesammelten access log files von proxycache.cs.uni-magdeburg.de (ca. 31 GB) und www.cache-uni-magdeburg.de (ca. 3 GB) bieten hervorragende Möglichkeiten für ein weiteres, detailliertes data mining und Gewinnung neuer Erkenntnisse in Hinsicht auf verbessertes, kooperatives Proxycaching und auch der Gestaltung von WWW-Inhalten sowie Administration von WWW-Servern an sich. Nur durch weitere Forschungen und Entwicklung neuer Methoden wird es möglich sein, den Proxycache-Dienst nicht nur für ISP, sondern auch für *alle* Nutzer attraktiv und absolut transparent zu machen.

5 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde ausgehend von der Analyse verschiedener Statistiken und Erfahrungen dargestellt, daß sich das World Wide Web aufgrund seiner Attraktivität nicht nur für Wissenschaft und Technik, sondern auch in den Bereichen Unterhaltung und E-Commerce zu einem sehr populären Medium entwickelt und zu einem sehr hohen Bedarf an Bandbreite und Server-Ressourcen geführt hat. Da diese oftmals nicht ausreichend zur Verfügung steht, führt das erfahrungsgemäß sehr oft zu einer Verminderung der Dienstgüte von WWW-Diensten. Aus dieser Motivation heraus wurden Möglichkeiten zur Verhinderung eines allgemeinen World Wide Wait aufgezeigt und anschließend auf die derzeit wohl am vielversprechenste, dem Proxycaching, detailliert eingegangen.

Dabei wurde dem Leser ein Einblick in die Client-Server-Architektur von Internet-Diensten gegeben, der Begriff Proxycache sowie dessen Funktionsweise beschrieben und die Notwendigkeit von Proxycache-Verbunden in Form einer Cache-Hierarchie, aber auch die Möglichkeit der dadurch entstehenden Probleme und eventuelle Lösungen erläutert. Einen sehr wichtigen Teil stellt dabei die Darstellung der Ergebnisse der Recherchen des Autors zum Bereich der Proxycache- Kommunikations- und Routingprotokolle dar, die bisher in dieser kompakten Form von niemandem weltweit veröffentlicht wurden.

Desweiteren wurde kurz auf verschiedene Proxy[cache]-Software eingegangen und eine diesbezügliche Auswertung von Logdateien von WWW-Servern der FIN vorgenommen, sowie herausgestellt, warum gerade die Proxycache-Software Squid zur Etablierung einer Proxycache-Hierarchie an der Otto-von-Guericke-Universität Magdeburg ausgewählt wurde.

Anschließend wurde sich ausgiebig mit dem Thema Performance-Tuning von Proxycaches und Konsistenz von Cache-Objekten beschäftigt. Dabei wurde auf die wichtigsten Konfigurationsparameter von Squid und dessen Anforderungen eingegangen,

Vorschläge für eine ausgewogene Konfiguration eines Proxycaches unterbreitet und Probleme bzgl. Hard- und Software aufgezeigt. Ebenso wurden aus den von Squid benutzten Regeln für das Cachen von WWW-Objekten sowie entsprechenden RFCs Einflüsse von WWW-Servern und User Agents auf die Performance von Proxycaches aufgezeigt und Empfehlungen für die Administration von WWW-Servern und die gestaltung von WWW-Seiten gegeben, sowie Maßnahmen zur Verbesserung von Trefferquoten durch den Einsatz von Redirectors aufgezeigt.

Zusätzlich wurde kritisiert, daß es bis dato nur befriedigende Werkzeuge zum Cache-Monitoring existieren, diese aber nicht zur detaillierten Analyse ausreichen. Darauf hin wurde ein Modell für ein neues, noch zu implementierendes Werkzeug entwickelt und dessen Vor- bzw. Nachteile gegenüber dem derzeit eingesetzten Programmen zur Überwachung der Proxycaches der Magdeburger Cache-Hierarchie diskutiert. Notwendigerweise wurden ebenfalls Empfehlungen für die wichtigsten, zu überwachenden Daten von Squid offeriert.

Der abschließende Teil der Arbeit befaßt sich mit der Auswertung bisher bekannter und veröffentlichter Verfahren zum Lastausgleich bzgl. WWW-Client und Proxycaches als auch zwischen Proxycaches selbst auf Basis von Langzeitanalysen der im Wirkbetrieb von Proxycaches erstellten „access_log“-Dateien. Da die Ergebnisse durchgeführter Untersuchungen zum load balancing zwischen verschiedenen Proxycaches nicht sonderlich befriedigen ausfallen, wird durch die Untersuchung eigener Verfahren versucht, mögliche Wege für neue „load balancing“-Algorithmen aufzuzeigen und zu weiteren Forschungen in dieser Richtung zu motivieren, denn diese stellen meiner Meinung nach in Zukunft eine Schlüsselposition dar, um auch die Bedürfnisse der Nutzer noch besser befriedigen zu können.

Hinsichtlich der Magdeburger Cache-Hierarchie ist zu sagen, daß die derzeitige hard- und softwaremäßige Konfiguration der Proxycaches ausreichend ist, jedoch eine Aufrüstung bzgl. Hardware (schnellere Platten und -Controler, mehr Hauptspeicher) und beim Einsatz der Software Squid in der Version >= 2.x evtl. auch mehr Rechenleistung notwendig wird, sowie alle Nutzer der Universität diese Proxycaches nutzen. Spätestens dann sollte man sich über den Einsatz eines Lastausgleichsverfahrens einig sein und dieses bis zu den WWW-Clients durch den Einsatz geeigneter Scripte zur automatischen Proxy-Konfiguration konsequent durchsetzen. Des weiteren sollte immer wie-

der darauf hingewiesen werden, daß durch den Einsatz von Proxycaches Zeit und damit auch Geld gespart werden kann. Eine regelmäßige Ankündigung in den Newsgruppen der Universität Magdeburg wäre ratsam und die zentrale Administration der Proxycache-Einstellungen (z.B. in Form von periodisch ablaufenden Scripten oder Programmen) aller Nutzer wünschenswert. Grundsätzlich ist darauf zu achten, daß möglichst viele Nutzer von den Proxycaches gebrauch machen, um einen möglichst hohen Wirkungsgrad hinsichtlich Trefferquote und eingesparter Bandbreite erzielen zu können.

Der Proxycache-Markt sollte weiterhin beobachtet werden, um rechtzeitig bei Notwendigkeit auf neue, innovative Produkte umsteigen zu können, soweit diese verfügbar sind. Daß neue Produkte entwickelt bzw. bisherige intensiv weiterentwickelt werden, steht außer Frage, denn schon jetzt haben namenhafte Firmen wie CISCO, Network Appliance, Inktomi erkannt, daß es hier einen sehr starken Bedarf an solchen Produkten gibt und Proxycache-Software bereits auch von fast allen namenhaften ISP eingesetzt wird (z.B. AOL, Deutsche Telekom AG, UU Net).

Anhang A

A.1 Anzahl der im DNS registrierten Maschinen

Die Anzahl der im Domain Name Service (DNS) registrierten Maschinen gilt allgemein als Indikator für die tatsächliche Anzahl der mit dem Internet verbundenen Maschinen. Tabelle A.1 stellt die in [3] ermittelte Anzahl der im DNS registrierten Hosts sowie die Zuwachsrate der registrierten Hosts gegenüber dem Vorjahr dar.

Zeitpunkt der Messung	**Jan. '93**	**Jan. '94**	**Jan. '95**	**Jan. '96**	**Jan. '97**	**Jan. '98**
Anzahl der registrierten Hosts	1313000	2217000	5846000	14352000	21819000	29670000
Zuwachsrate gegenüber dem Vorjahr	???	68.8%	163.7%	245.5%	52.0%	36.0%

Tabelle A.1: Anzahl der im DNS registrierten Maschinen

Daraus ist ersichtlich, das es speziell in den Jahren 1995 und 1996 einen explosionsartigen Anstieg der im DNS registrierten Maschinen gekommen ist und diese Jahre mit dem Durchbruch des Internet zum Massenmedium einstufbar sind.

A.2 Regressionsanalyse zu NFSNET Statistiken

Alle zur Verfügung stehenden und benutzten Daten bzgl. NFSNET wurden mit Hilfe der Regressionsanalyse mit dem Ziel einer Trendbestimmung analysiert. Da alle Kurven für den Datenverkehr bzgl. der wichtigsten Protokolle einen exponentiellen Verlauf haben, wurden die Werte (transferierten Bytes) für die entsprechenden Meßpunkte (Monate[1]) logarithmiert und darauf eine lineare Regression angewandt.

Laut [7] ergeben sich damit folgende Lösungen für die Bestimmung der konkreten empirischen Parameter einer konkreten empirischen Regressionsgerade der Form $y = ax + b$:

1. Den ersten Meßpunkt stellt der Monat Dezember '92 dar. Ihm wurde der Wert 1 zugeordnet. Jedem folgenden Monat wurde der Wert des vorherigen Monats um 1 inkrementiert zugeordnet.

Gleichung A.1: Regressionskoeffizient:

$$a = \frac{\sum_{i=1}^{n} x_i y_i - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y_i\right)}{\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2}$$

Gleichung A.2: Ordinatenschnittpunkt:

$$b = \frac{1}{n}\left(\sum_{i=1}^{n} y_i\right) - \frac{a}{n}\left(\sum_{i=1}^{n} x_i\right)$$

Gleichung A.3: Restvarianz:

$$s_R^2 = \frac{1}{n-2}\left[\sum_{i=1}^{n} y_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} y_i\right)^2 - a\left(\sum_{i=1}^{n} x_i y_i - \frac{1}{n}\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i\right)\right]$$

Gleichung A.4: Streuung für a:

$$d_a = \sqrt{\frac{s_R^2}{\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2}}$$

Gleichung A.5: Streuung für b:

$$d_b = \sqrt{s_R^2\left(\frac{1}{n} + \frac{\frac{1}{n}\left(\sum_{i=1}^{n} y_i\right)^2}{\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2}\right)}$$

Da die Analyse mit den logarithmierten Werten der transferierten Bytes vorgenommen wurde, folgen für die konkreten empirischen Werte an transferierten Bytes:

Gleichung A.6: $\ln(bytes) = ax + b$

Gleichung A.7: $bytes = e^{ax+b}$

Gleichung A.8: $bytes = e^{b} * e^{ax}$

Daraus erhält man dann die konkrete empirische, monatliche Zuwachsrate an transferierten Bytes:

Gleichung A.9: Zuwachsrate / Monat = e^{a}

Gleichung A.10: mit einer Abweichung von e^{da} .

Monat	**ftp [MB]**	**http [MB]**	**nntp [MB]**	**smtp [MB]**	**telnet [MB]**	**gopher [MB]**	**irc [MB]**
13	3878.81	209.960	923.005	564.797	499.360	288.421	123.789
14	3951.70	250.646	987.858	613.502	560.618	348.949	133.796
15	4174.50	323.638	1036.68	713.516	602.789	368.865	153.896
16	4817.14	482.503	1213.75	861.118	660.597	447.677	181.401
17	4794.09	625.802	1317.49	919.230	664.413	482.076	199.700
18	5256.27	744.278	1430.93	918.786	667.826	517.543	194.251
19	5035.44	881.533	1553.99	920.166	691.529	528.506	180.057
20	5019.77	983.552	1636.24	866.403	720.546	516.966	190.889
21	5428.84	1221.73	1775.86	935.713	747.776	607.079	218.314
22	5632.25	1484.03	1943.22	1094.59	807.897	699.847	235.971
23	6547.60	2005.10	2076.94	1313.29	905.776	804.904	283.251
24	6640.67	2911.50	2164.45	1285.86	906.649	807.497	309.819
25	6413.95	3236.69	2215.47	1136.37	791.945	724.84	278.665
Regressionskoeff. lt. Gleichung A.1:	0.044	0.225	0.078	0.061	0.042	0.081	0.068
+- Abweichung lt. Gleichung A.4:	0.004	0.007	0.003	0.008	0.005	0.006	0.007
Ord.-schnittpkt. lt. Gleichung A.2:	7.697	2.465	5.839	5.659	5.746	4.733	3.994
+- Abweichung lt. Gleichung A.5:	0.072	0.136	0.054	0.148	0.088	0.123	0.129
Restvarianz lt. Gleichung A.3:	0.050	0.095	0.037	0.103	0.061	0.086	0.090
mon. Zuwachsrate lt. Gleichung A.9:	4.5%	25.2%	8.1%	6.3%	4.3%	8.4%	7.1%
+- Abweichung lt. Gleichung A.10:	0.37%	0.71%	0.28%	0.77%	0.46%	0.64%	0.67%

Tabelle A.2: Regressionsanalysedaten zum Datenverkehr im NSFNET

Die Regressionsanalyse wurde anhand der von Merit im NSFNET gesammelten Daten Daten [5] für den Zeitraum Dez. ‘93 bis Dez. ‘94 einschließlich vorgenommen, da ab Januar ‘95 bereits die Migrierung des Traffics in das neue NFS-Net begann. Der Regressionskoeffiezient und dessen Abweichung, der Ordinatenschnittpunkt und dessen Abweichung, sowie die Restvarianz in Tabelle A.2 beziehen sich auf die logarithmierten Werte der Transfervolumen der Monate 13 - 25, d.h. Dez. ‘94 - Dez. ‘95 inklusive.

Anhand der Berechnungen ist zu sehen, daß das Volumen an transferierten Bytes via http bedeutend schneller wächst, als das aller anderen Protokolle (in der Tabelle wurden die relevante Protokolle ausgewählt, die in den letzten Monaten des NSFNET das größte transferierte Datenvolumen aufweisen). Daraus kann abgleitet werden, daß spätestens ab Mai ‘95 das via http transferierte Datenvolumen das aller anderen Protokolle überstiegen hat und somit in Zukunft eine der größten Herausforderung an das Internet hinsichtlich der zu bewältigenden Datenvolumen darstellt.

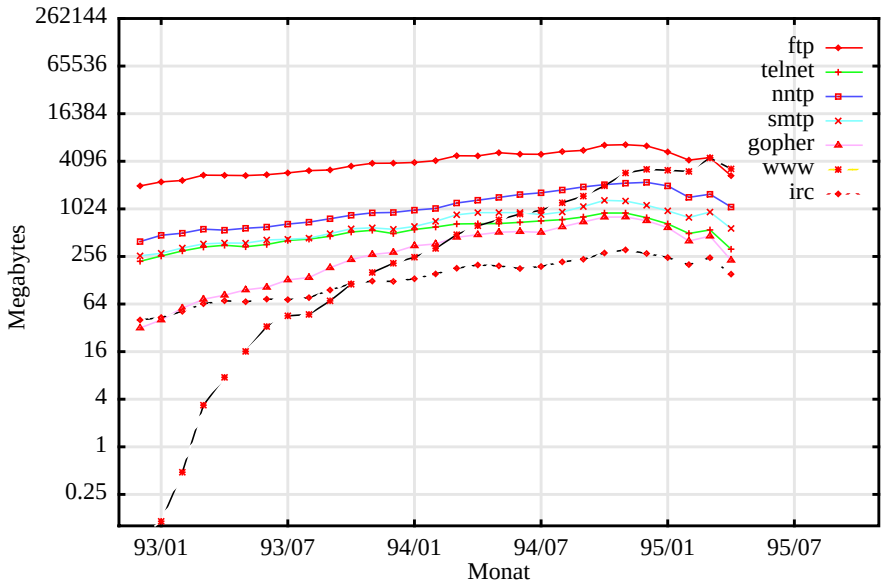

Abbildung A.1: Datenvolumen wichtiger Protokolle im NSFNET

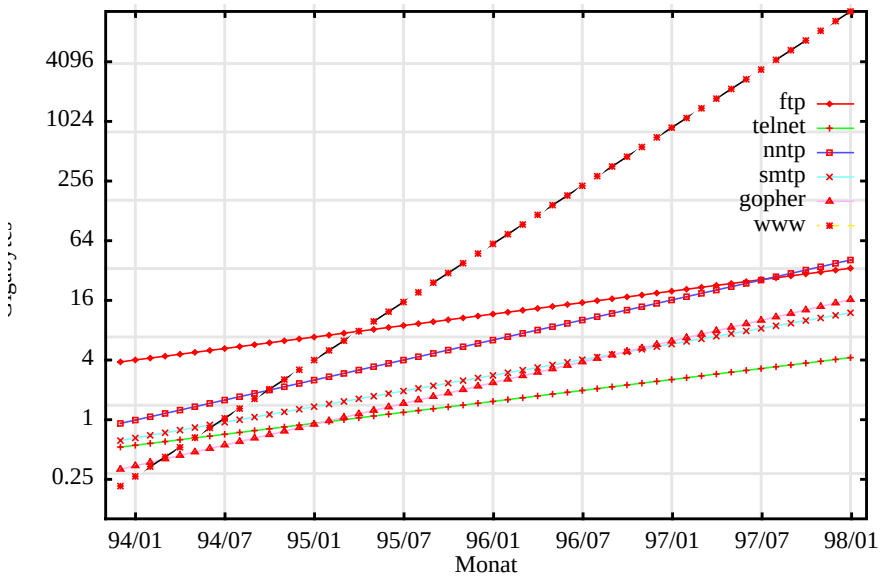

Abbildung A.2: Trend bzgl. Datenvolumen wichtiger Protokolle im NSFNET

A.3 Das Hypertext-Transfer-Protokoll

Das Hypertext-Transfer-Protokoll (HTTP) ist ein Client-Server Protokoll. Eine HTTP-Transaktion beginnt, indem der Client (z.B. ein WWW-Browser) eine Verbindung zum HTTP-Server aufbaut und eine Anfrage für ein Objekt an selbigen sendet, welche den in Tabelle A.3 schematisch dargestellten Aufbau haben (siehe auch [9] und [10]).

Anfrage	**möglicher Inhalt**	**Beispiel**
request-line	Method abs_path HTTP / Version mit Method = OPTIONS | GET | HEAD | POST | PUT | DELETE | TRACE	GET / index.shtml HTTP / 1.0
general-header	Cache-Control | Connection | Date | Pragma | Transfer-Encoding | Upgrade | Via	Pragma: no-cache
request-header	Accept | Accept-Charset | Accept-Encoding | Accept-Language | Authorization | From | Host | If-Modified-Since | If-Match | If-None-Match | If-Range | If-Unmodified-Since | Max-Forwards | Proxy-Authorization | Range | Referer | User-Agent	Host: ivs.cs.uni-magdeburg.de User-Agent: GetUrl 1.0 Accept: */*
entity-header	Allow | Content-Base | Content-Encoding | Content-Language | Content-Length | Content-Location | Content-MD5 | Content-Range | Content-Type | ETag | Expires | Last-Modified	
CRLF	Leerzeile, beendet durch CarrigeReturn und LineFeed	
message-body		

Tabelle A.3: Aufbau einer Client Anfrage

wobei der *request header* und *entity header* entfallen können. Im folgenden ist als Beispiel solch eine Anfrage vom WWW-Browser Netscape 3.01 dargestellt[1]:

```
GET /sw-eng/us/index.shtml HTTP/1.0                                  request-line
Connection: Keep-Alive                                               general-header line
User-Agent: Mozilla/3.01Gold (X11; I; SunOS 5.5.1 sun4u)             request-header line
Host: ivs.cs.uni-magdeburg.de                                        request-header line
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*     request-header line
                                                                     Leerzeile (CRLF)
```

1. Zu Testzwecken kann man auch eine Verbindung zum HTTP-Server unter Zuhilfenahme des Programms *telnet* (z.B. telnet ivs.cs.uni-magdeburg.de 80) aufbauen und die gesamte Anfrage interaktiv im Klartext eingeben.

Der HTTP-Server sendet das Ergebnis der Anfrage entsprechend dem in Tabelle A.4 dargestellten schematischen Aufbau, wobei wiederum nur die *status-line*[1] sowie die Leerzeile (*CRLF*) erforderlich sind.

Antwort	**möglicher Inhalt**	**Beispiel**
status-line	HTTP / Version Status-Code Reason-Phrase mit Status = {100..101 | 200..206 | 300..305 | 400..414 | 500..505} und mit Reason-Phrase = Text außer CRLF	HTTP / 1.1 200 OK
general-header	Cache-Control | Connection | Date | Pragma | Transfer-Encoding | Upgrade | Via	Date: Sun, 27 Jul 1997 21:12:30 GMT
response-header	Age | Location | Proxy-Authenticate | Public | Retry-After | Server | Vary | Warning | WWW-Authenticate	Server: Apache / 1.2b10
entity-header	Allow | Content-Base | Content-Encoding | Content-Language | Content-Length | Content-Location | Content-MD5 | Content-Range | Content-Type | ETag | Expires | Last-Modified	Last-Modified: Fri, 25 Jul 1997 11:24:05 GMT ETag: „14e954-b9a-33d88cd5“
CRLF	Leerzeile, beendet durch CarrigeReturn und LineFeed	
message-body		<HTML>.<HEAD><TITLE>Software Metrics - Forum</TITLE> ...

Tabelle A.4: Aufbau einer Antwort eines HTTP-Servers

Der *message-body* enthält das eigentliche, unveränderte Objekt, nach dem angefragt wurde. Der folgende Abschnitt zeigt das Ergebnis der Anfrage vom ersten Beispiel:

```
HTTP/1.1 200 OK                                          status-line
Date: Sun, 27 Jul 1997 21:12:30 GMT                      general-header line
Server: Apache /1.2b10                                   response-header line
Last-Modified: Fri, 25 Jul 1997 11:24:05 GMT             entity-header line
ETag: „14e954-b9a-33d88cd5“
Connection: close                                        general-header line
                                                         Leerzeile (CRLF)
<HTML>.<HEAD><TITLE>Software Metrics - Forum</TITLE> ... body
```

A.4 Uniform Resource Locators

Ein WWW-Objekt wird durch seinen Uniform Resource Locator (URL) identifiziert. Aufgrund dieser URL weiß der WWW-Client, zu welchem Server er eine Verbindung

1. Auf die Übersetzung der entsprechenden Begriffe ins Deutsche wurde absichtlich verzichtet um evtl. Mißverständnisse bzgl. der entsprechenden RFCs auszuschließen.

über welches Protokoll aufbauen muß und welches Objekt abgefragt werden soll. Verallgemeinert haben URLs die folgende Form *<scheme>:<scheme-specific-part>*. Tabelle A.5 gibt einen Überblick über die gebräuchlichsten URLs (die genaue Syntax und Semantik kann in [8] nachgelesen werden).

scheme	**scheme-specific-part**	**Bemerkung**
ftp	//<user>:<password>@<host>:<port>/<cwd1>/<cwd2>/.../<cwdN>/<name>;type=<typecode>	File Transfer protocol
http	//<host>:<port>/<path>?<searchpart>	Hypertext Transfer Protocol
gopher	//<host>:<port>/<gopher-path>	The Gopher protocol
mailto	<rfc822-addr-spec>	Electronic mail address
news	<newsgroup-name> oder <message-id>	USENET news
nntp	//<host>:<port>/<newsgroup-name>/<article-number>	USENET news using NNTP access
telnet	//<user>:<password>@<host>:<port>/	Reference to interactive sessions
wais	//<host>:<port>/<database> oder //<host>:<port>/<database>?<search> oder //<host>:<port>/<database>/<wtype>/<wpath>	Wide Area Information Servers
file	//<host>/<directory>/<directory>/.../<name>	Host-specific file names
pro-spero	//<host>:<port>/<hsoname>;<field>=<value>	Prospero Directory Service

Tabelle A.5: Verschiedene Formen von URL's

A.5 User Agent Statistiken

Die Analyse der Zugriffe auf die WWW-Server der Fäkultät für Informatik der Otto-von-Guericke-Universität Magdeburg bzgl. WWW-Browser zeigt, daß in der Summe zwischen 80-90% aller an diese WWW-Server gestellten Anfragen durch WWW-Browser erfolgte. Wie in Abbildung A.3 zu sehen ist, kann dies sogar noch deutlicher eingeschränkt werden, denn diese 80-90% sind i.d.R. vom Netscape Navigator/Communicator (Mozilla) oder Microsoft Internet Explorer (MSIE) gestellte Anfragen. Die Anteile aller anderen WWW-Browser liegt i.d.R. deutlich unter 5%, weshalb diese nicht in Abbildung aufgenommen wurden. Eine Ausnahme scheint jedoch http://java.cs.uni-magdeburg.de zu bilden. Hier liegt die Summe aller Zugriffe via Mozilla und MSIE nur zwischen 70-80%. Berücksichtigt man, daß auf diesem Server ein relativ

großer Anteil der Anfragen (10-20%) von Suchmaschinen und WWW-Spiegel-Programmen verursacht werden, trifft auch hier die Aussage zu, daß ca. 80-90% aller WWW-Browser-Zugriffe durch Mozilla oder MSIE erfolgen. Generell wird angenommen, daß diese Aussage auch auf alle anderen, öffentlich zugänglichen WWW-Server zutrifft und somit die Faustregel gilt:

80-90% aller Anfragen auf einen öffentlich zugänglichen WWW-Server werden unter Zuhilfenahme der WWW-Browser Mozilla oder MSIE realisiert.

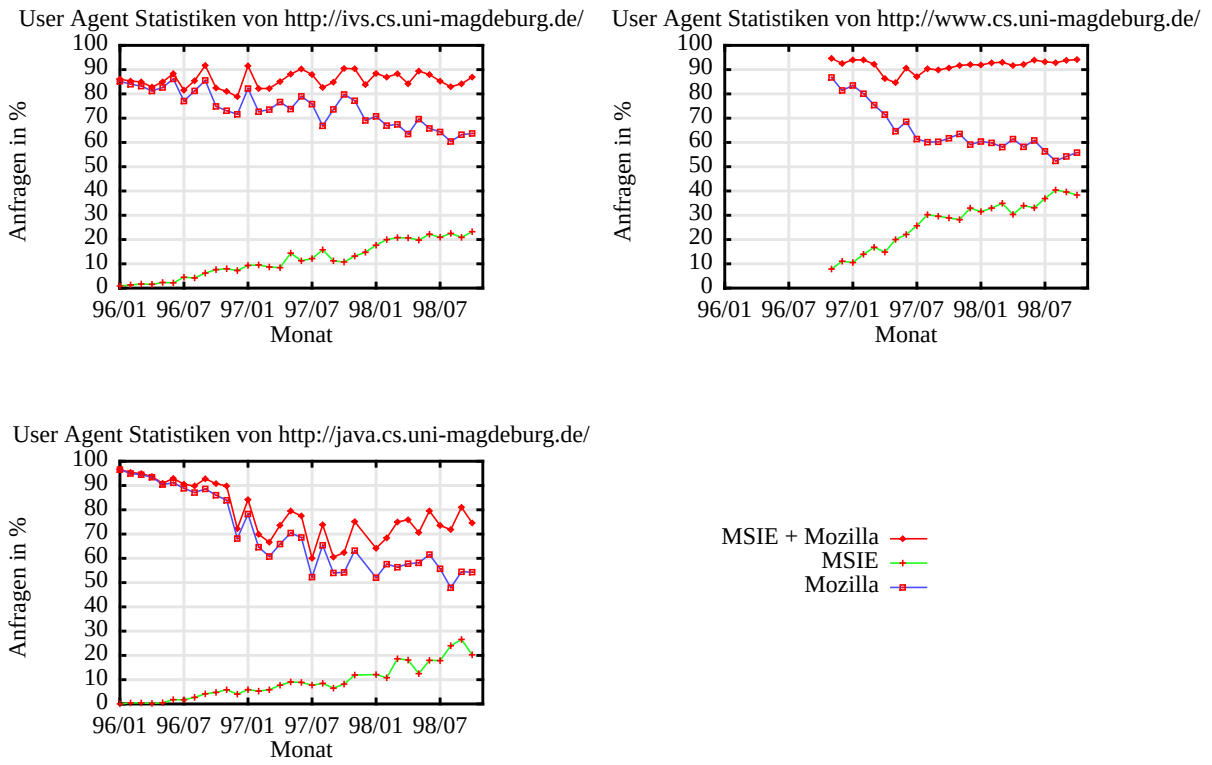

Abbildung A.3: User Agent Statistiken von WWW-Servern

Somit kann geschlossen werden, daß wahrscheinlich auch in einem Unternehmen oder einer Institution, in der die Nutzung von WWW-Browser-Software nicht auf ein bestimmtes Produkt eingeschränkt ist, 80-90% aller Anfragen an WWW-Server mittels Mozilla oder MSIE erfolgen. Es wird vermutet, daß selbiges auch für Zugriffe auf FTP- und Gopher-Server zutrifft, was jedoch auf grund der entsprechenden Protokolle nur sehr schlecht bzw. für externe Zugriffe auf den eigene Server nicht überprüft werden kann.

Anmerkung: Die Daten für diese Statistiken wurden durch die monatliche Analyse der User Agent Log-Dateien der entsprechenden Apache-WWW-Servers (agent_log) durch das eigens für diesen Zweck entwickelte Programm „browsercounter.pl“ extrahiert und als HTML-Dateien aufbereitet. Diese Dateien und somit die detailierten Ergebnisse der Analysen können unter folgenden URLs nachgelesen werden:

http://ivs.cs.uni-magdeburg.de/stats/WWW/
http://www.cs.uni-magdeburg.de/stats/WWW/
http://java.cs.uni-magdeburg.de/stats/WWW/

Informationen zu dem oben genannten Programm sind unter http://ivs.cs.uni-magdeburg.de/~elkner/webtools/browsercounter.shtml verfügbar.

A.6 Tests zum Lastenausgleich

Grundlage bilden die monatlich geführten „access_log“-Dateien des Parent Proxycaches der MCH (Proxycache.cs.uni-magdeburg.de) im Zeitraum von Januar 1998 bis einschließlich Oktober 1998. Um Verfälschungen der Ergebnisse zu vermeiden, wurden alle cache_object Anfragen sowie Anfragen von Neighbors und der Maschine, auf dem das PCMS läuft, bei den Analysen nicht berücksichtigt. Zeitraum jeder einzelnen Messung ist Monatsbeginn bis Monatsende. In Abbildung A.4 ... Abbildung A.23 ist die absolute Anzahl der Anfragen aller Clients an den jeweiligen Cache, sowie die an seine Clients übertragene Anzahl von Bytes im Intervall von 2 Stunden dargestellt (absolut ... Summe der Werte aller bisherigen Meßintervalle). Abbildung A.24 ... Abbildung A.43 sind analog, bis auf das dort alle Werte relativ dargestellt sind, d.h. sich immer nur auf das letzte Meßintervall (2 h) beziehen.

Alle benutzten Daten sowie erstellten Grafiken befinden sich auf der beigefügten CD unter dem Verzeichnis /data/results/testN/ mit N = Nummer des durchgeführten Tests und können bei Interesse für weitere bzw. detailliertere Auswertungen genutzt werden.

A.6.1 CARP (Test 1) und Squid-CARP (Test 2)

Proxycache Adresse	Loadfactor	Proxycache Adresse	Loadfactor
leipzig.www-cache.dfn.de	0.25	berlin.www-cache.dfn.de	0.25
nuernberg.www-cache.dfn.de	0.25	hamburg.www-cache.dfn.de	0.25

Tabelle A.6: LB CARP, Squid-CARP: Parameter

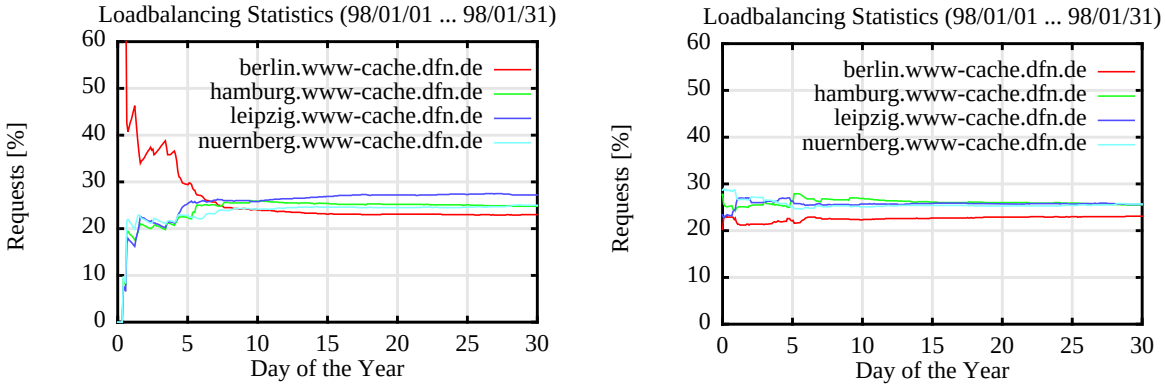

Abbildung A.4: LB (absolut): CARP (left), Squid-CARP (right); total: 724.091 requests

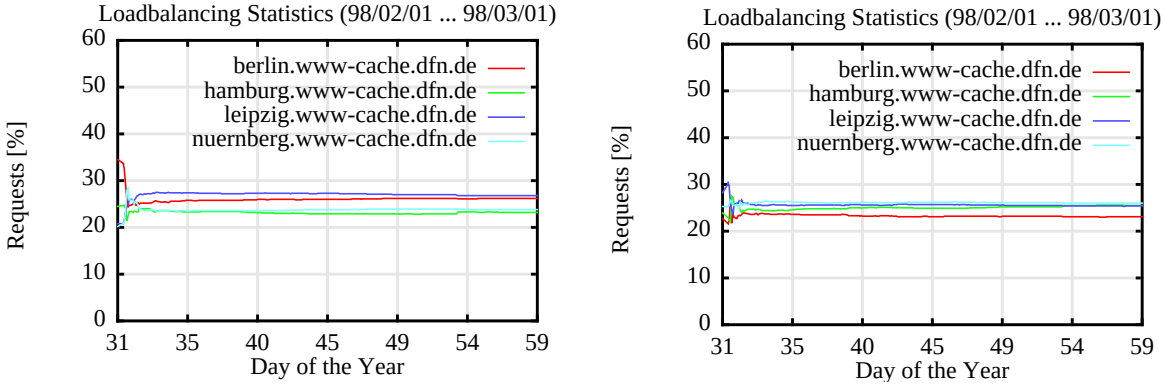

Abbildung A.5: LB (absolut): CARP (left), Squid-CARP (right); total: 813.652 requests

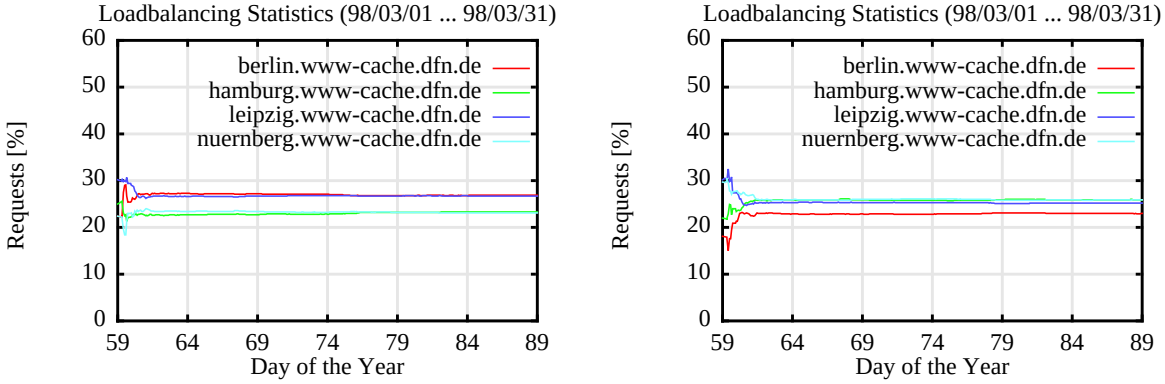

Abbildung A.6: LB (absolut): CARP (left), Squid-CARP (right); total: 797.286 requests

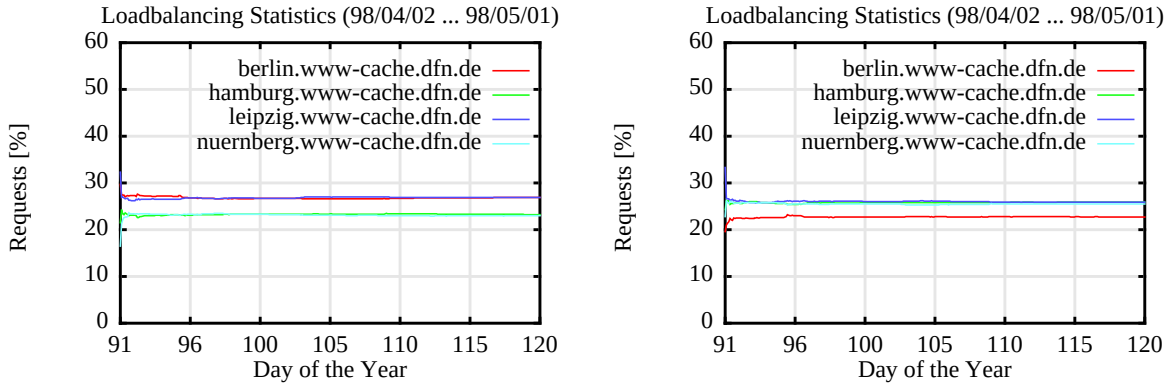

Abbildung A.7: LB (absolut): CARP (left), Squid-CARP (right); total: 1.012.006 requests

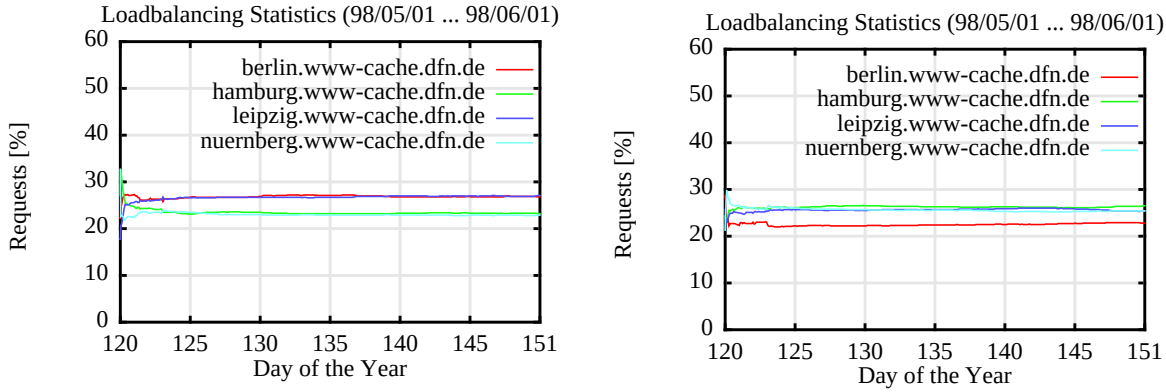

Abbildung A.8: LB (absolut): CARP (left), Squid-CARP (right); total: 1.065.412 requests

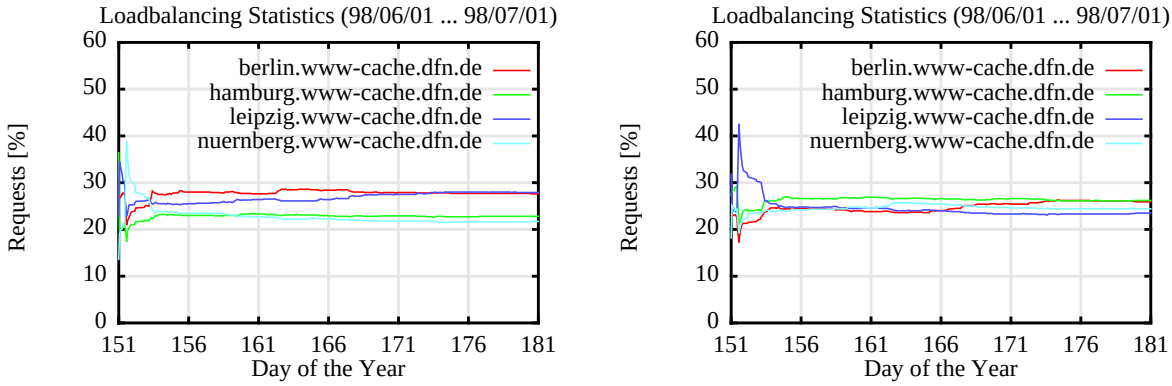

Abbildung A.9: LB (absolut): CARP (left), Squid-CARP (right); total: 1.202.550 requests

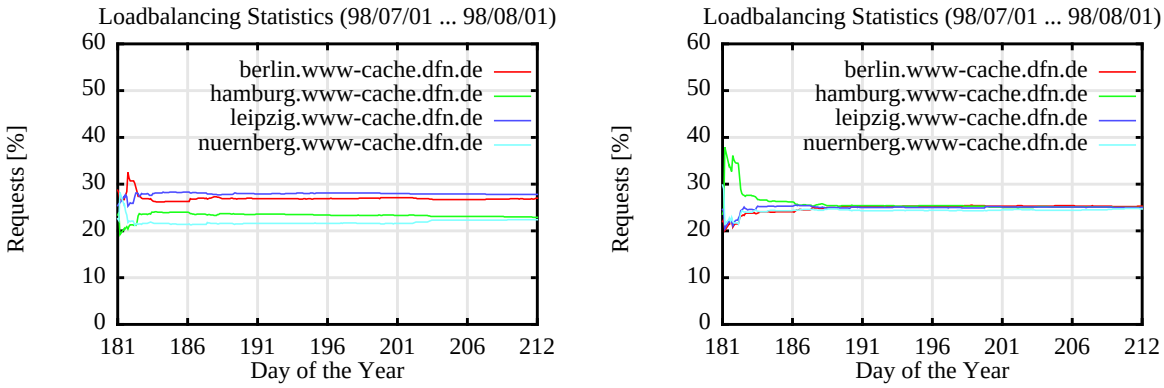

Abbildung A.10: LB (absolut): CARP (left), Squid-CARP (right); total: 1.083.031 requests

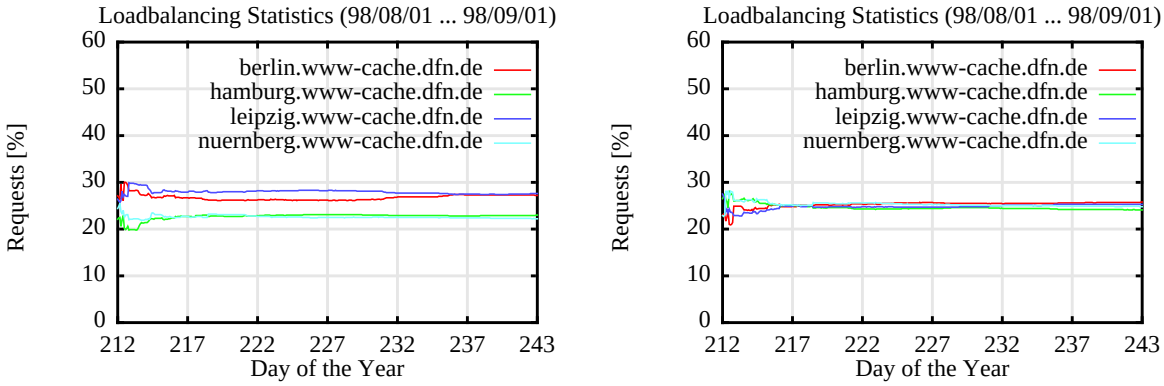

Abbildung A.11: LB (absolut): CARP (left), Squid-CARP (right); total: 947.991 requests

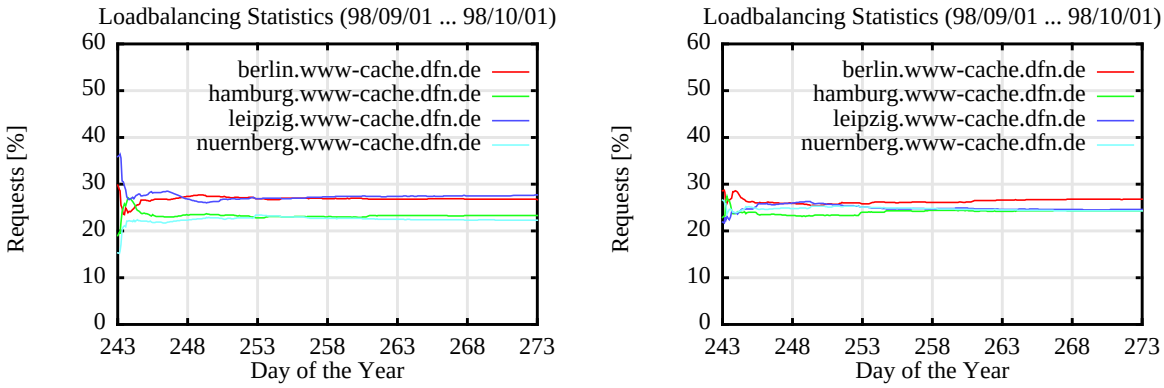

Abbildung A.12: LB (absolut): CARP (left), Squid-CARP (right); total: 960.227 requests

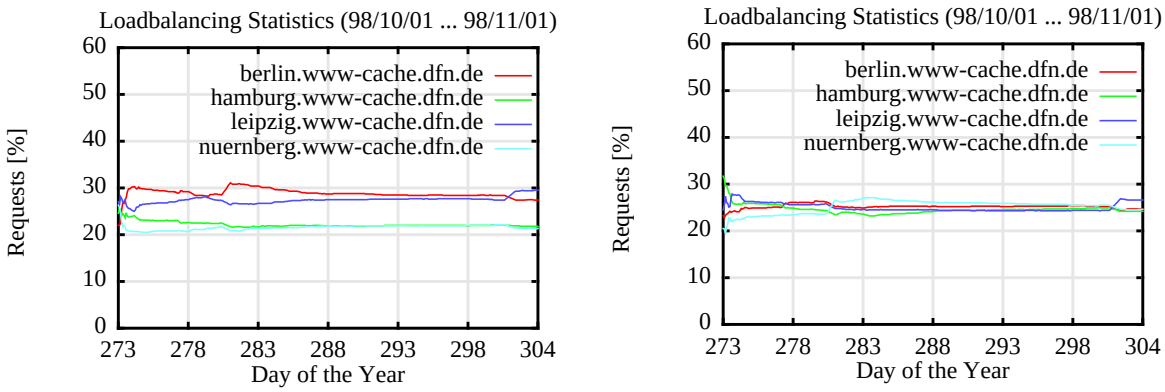

Abbildung A.13: LB (absolut): CARP (left), Squid-CARP (right); total: 1.257.210 requests

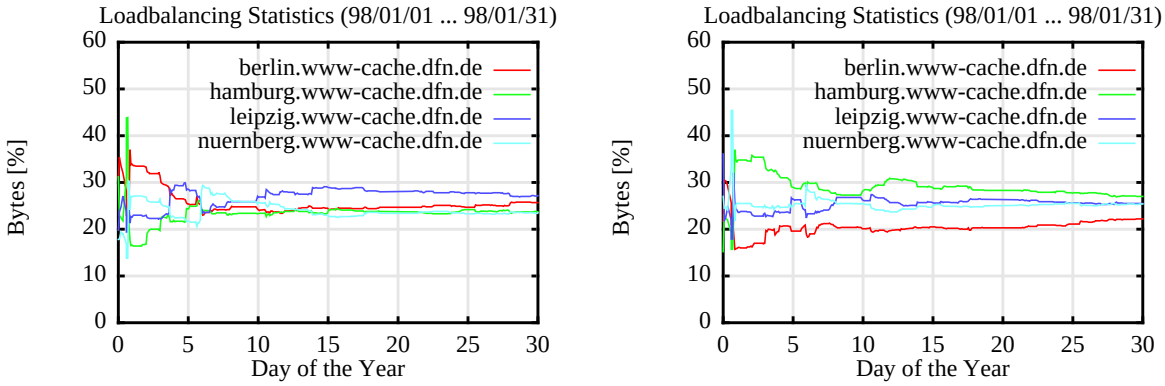

Abbildung A.14: LB (absolut): CARP (left), Squid-CARP (right); total: 8.222.591.572 bytes

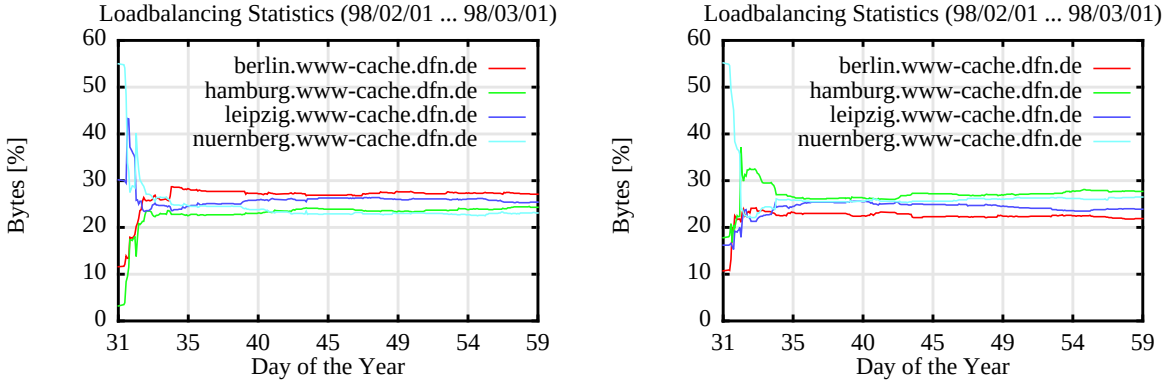

Abbildung A.15: LB (absolut): CARP (left), Squid-CARP (right); total: 8.720.394.922 bytes

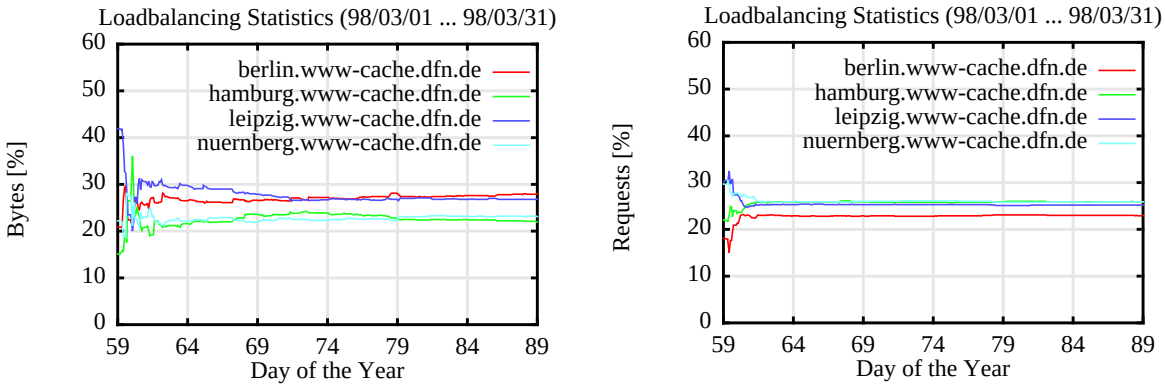

Abbildung A.16: LB (absolut): CARP (left), Squid-CARP (right); total: 10.801.065.979 bytes

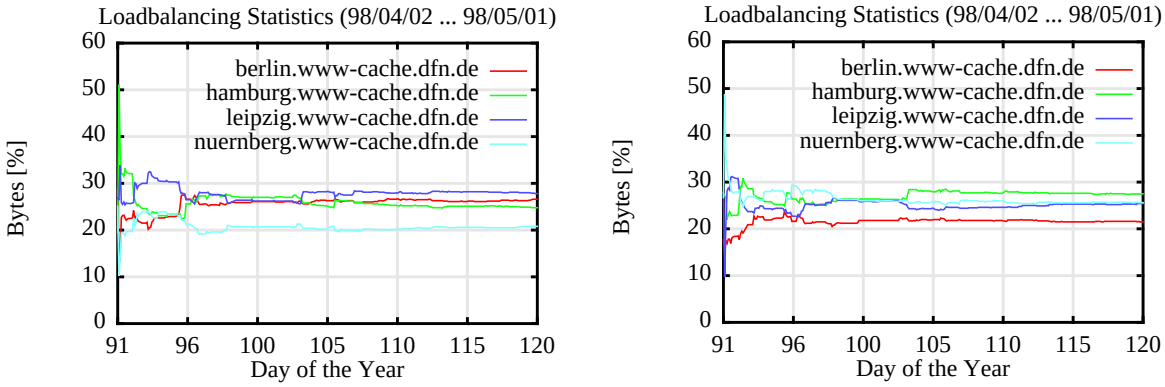

Abbildung A.17: LB (absolut): CARP (left), Squid-CARP (right); total: 12.688.800.455 bytes

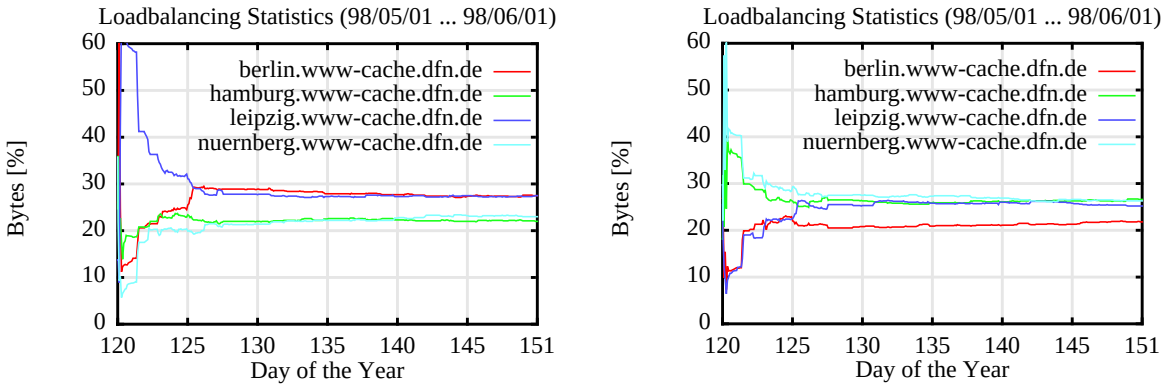

Abbildung A.18: LB (absolut): CARP (left), Squid-CARP (right); total: 12.427.657.633 bytes

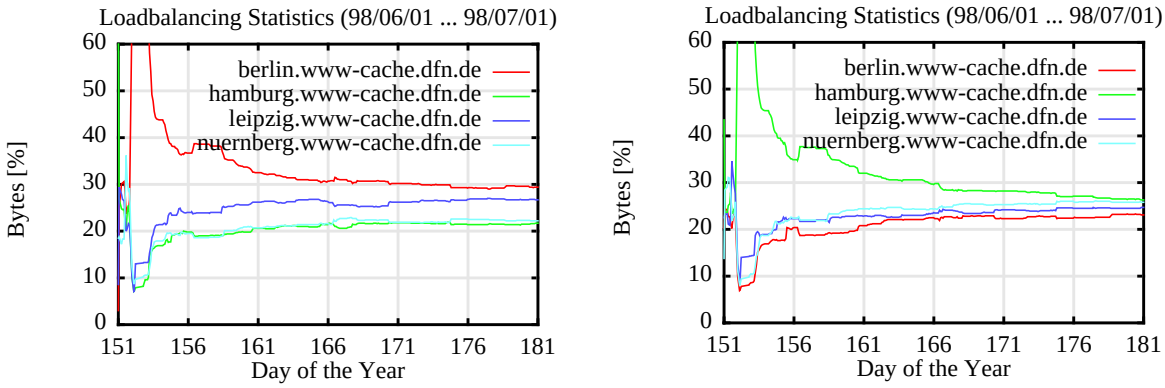

Abbildung A.19: LB (absolut): CARP (left), Squid-CARP (right); total: 11.719.732.474 bytes

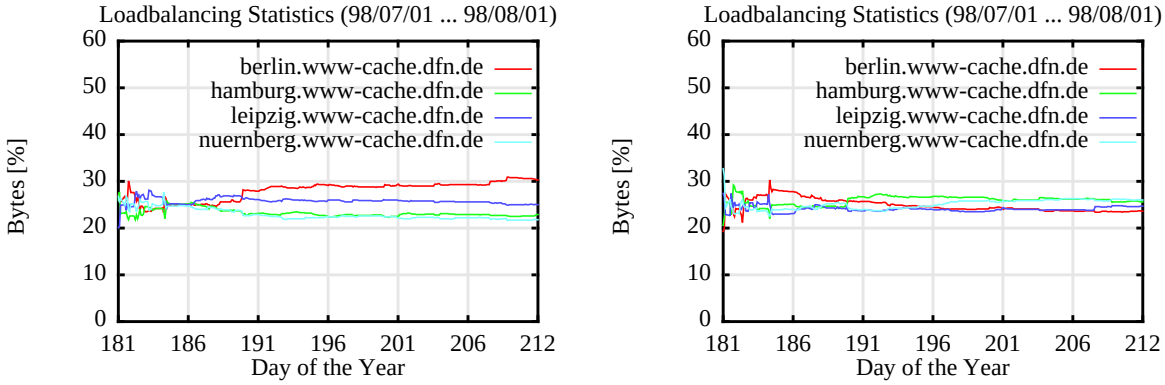

Abbildung A.20: LB (absolut): CARP (left), Squid-CARP (right); total: 11.630.194.766 bytes

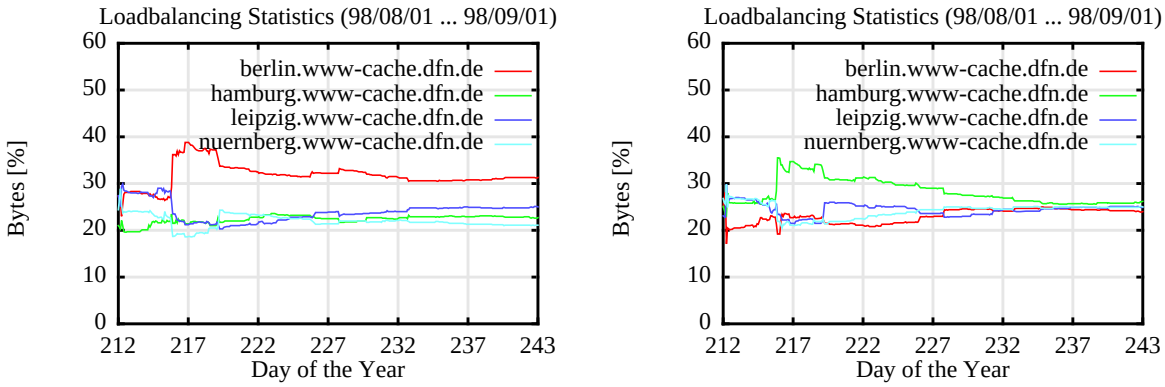

Abbildung A.21: LB (absolut): CARP (left), Squid-CARP (right); total: 14.309.593.596 bytes

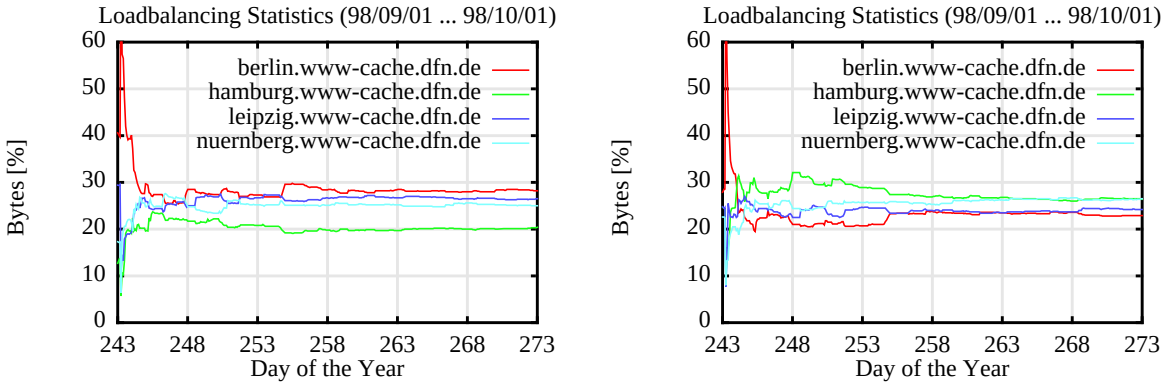

Abbildung A.22: LB (absolut): CARP (left), Squid-CARP (right); total: 11.359.838.505 bytes

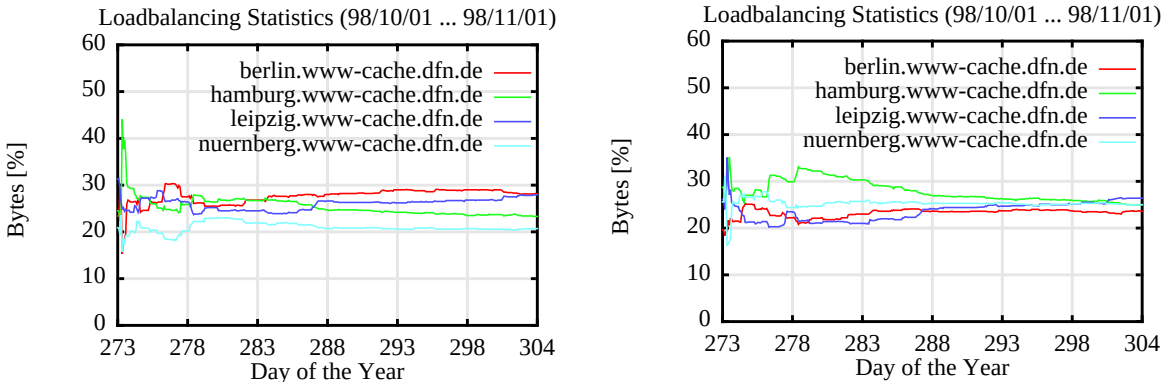

Abbildung A.23: LB (absolut): CARP (left), Squid-CARP (right); total: 13.249.527.918 bytes

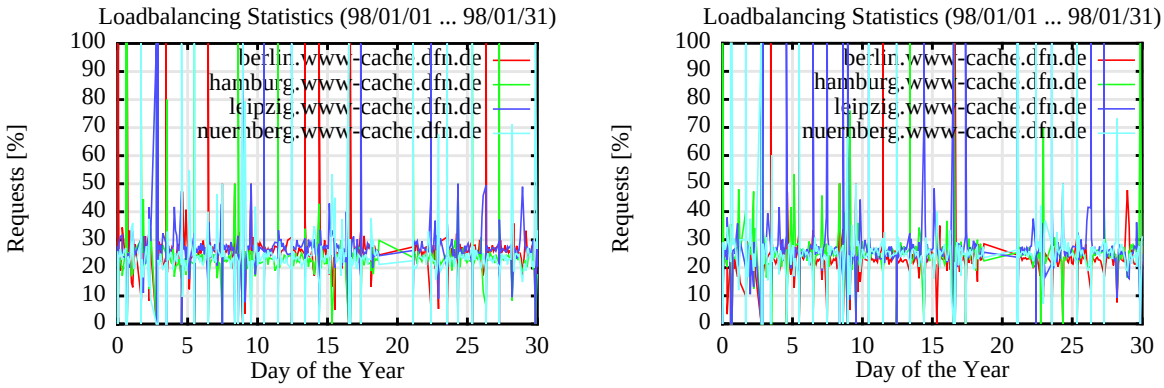

Abbildung A.24: LB (relativ): CARP (left), Squid-CARP (right); total: 724.091 requests

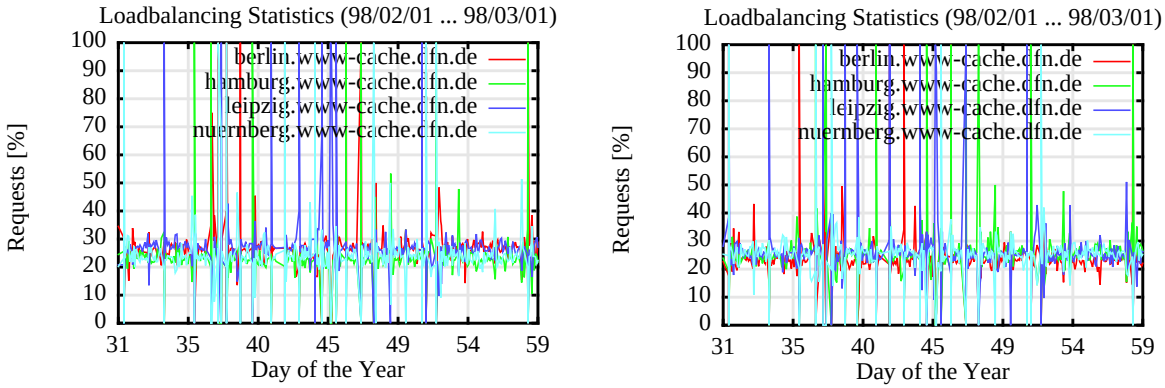

Abbildung A.25: LB (relativ): CARP (left), Squid-CARP (right); total: 813.652 requests

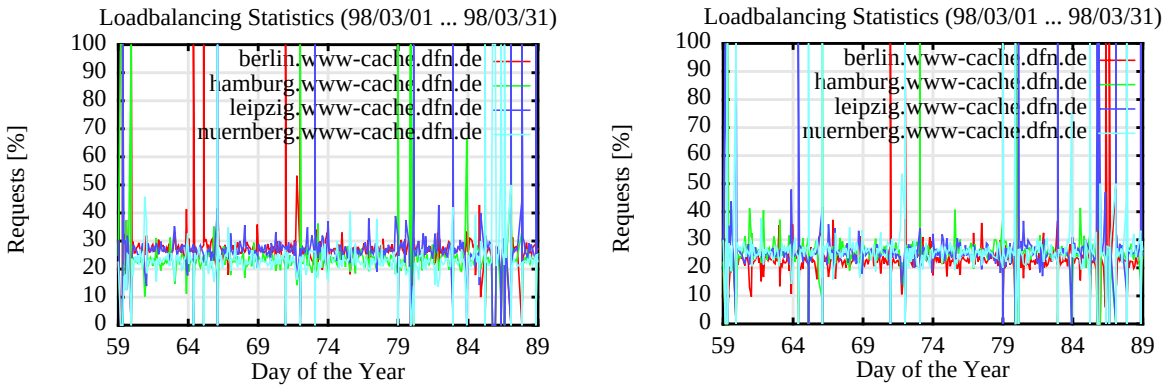

Abbildung A.26: LB (relativ): CARP (left), Squid-CARP (right); total: 797.286 requests

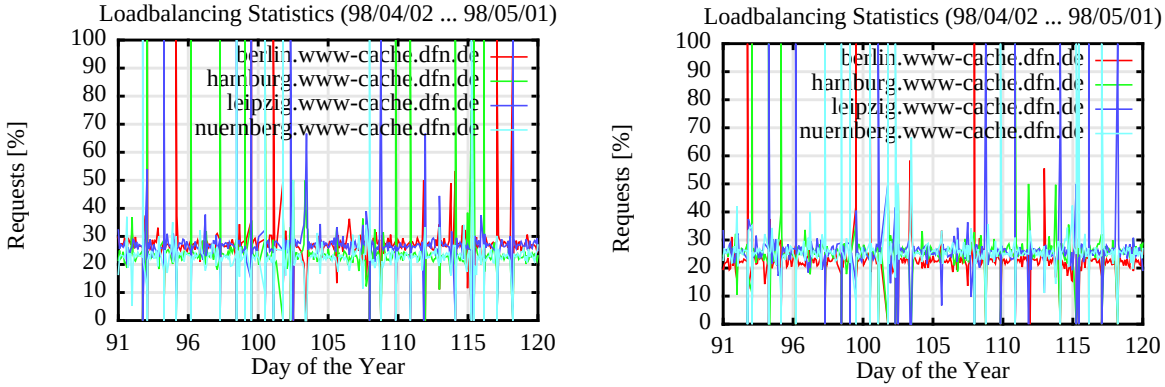

Abbildung A.27: LB (relativ): CARP (left), Squid-CARP (right); total: 1.012.006 requests

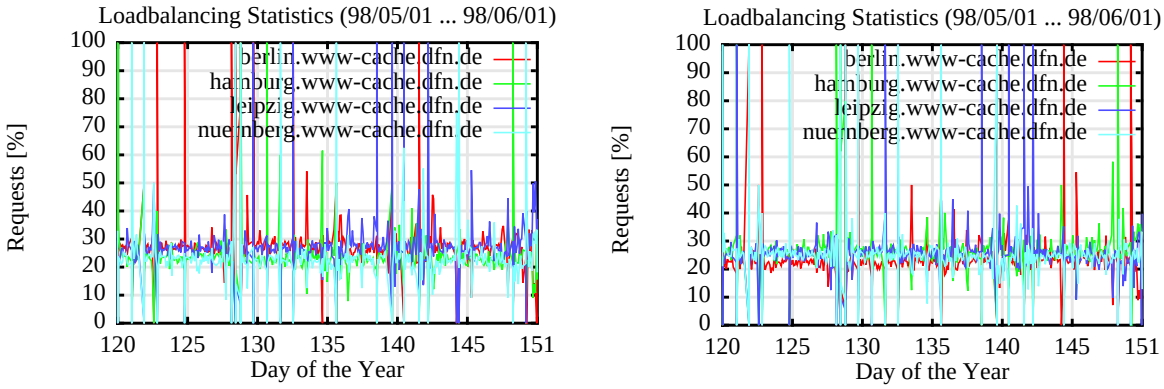

Abbildung A.28: LB (relativ): CARP (left), Squid-CARP (right); total: 1.065.412 requests

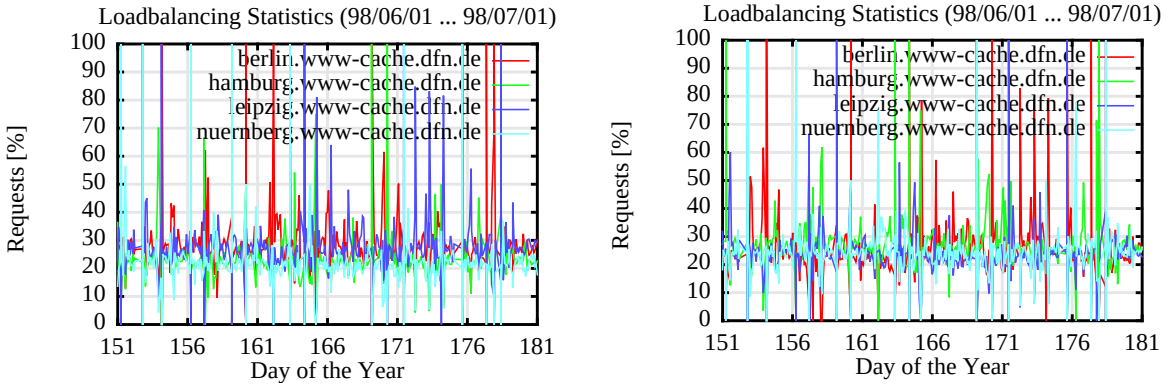

Abbildung A.29: LB (relativ): CARP (left), Squid-CARP (right); total: 1.202.550 requests

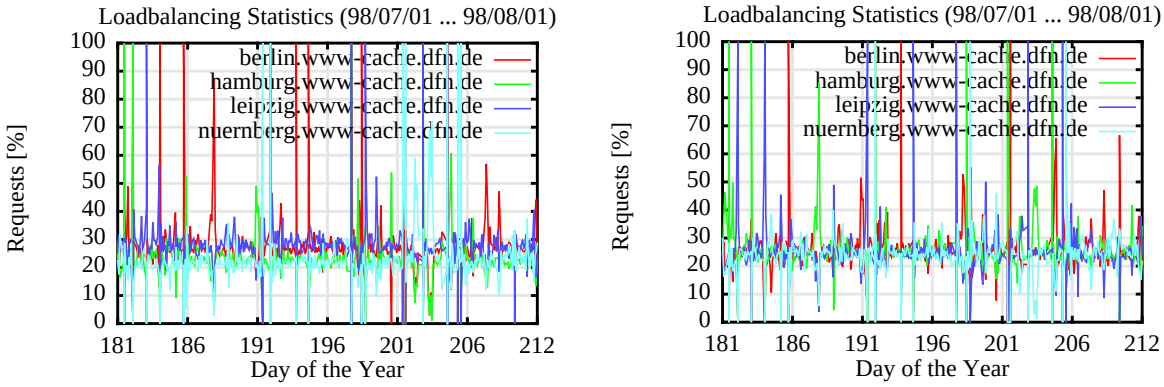

Abbildung A.30: LB (relativ): CARP (left), Squid-CARP (right); total: 1.083.031 requests

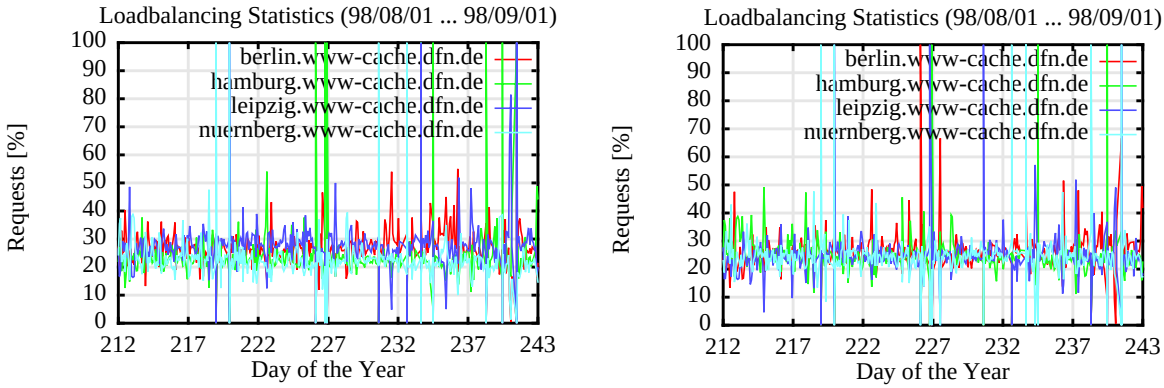

Abbildung A.31: LB (relativ): CARP (left), Squid-CARP (right); total: 947.991 requests

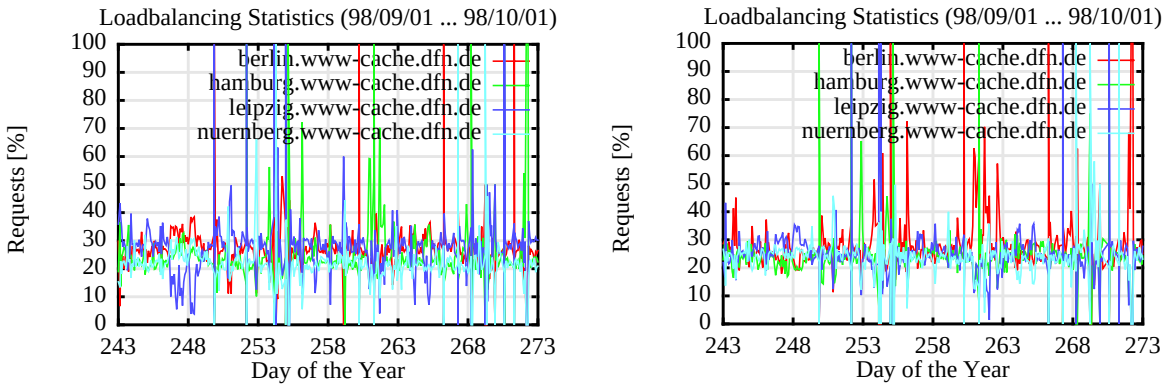

Abbildung A.32: LB (relativ): CARP (left), Squid-CARP (right); total: 960.227 requests

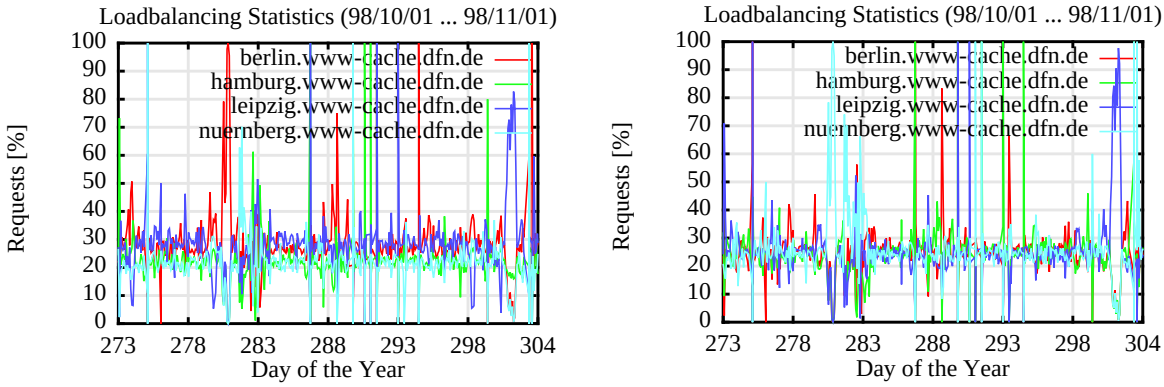

Abbildung A.33: LB (relativ): CARP (left), Squid-CARP (right); total: 1.257.210 requests

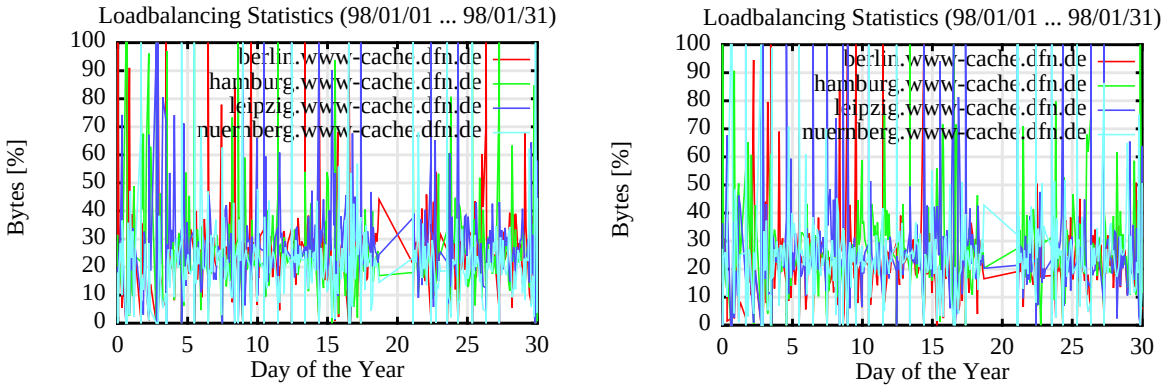

Abbildung A.34: LB (relativ): CARP (left), Squid-CARP (right); total: 8.222.591.572 bytes

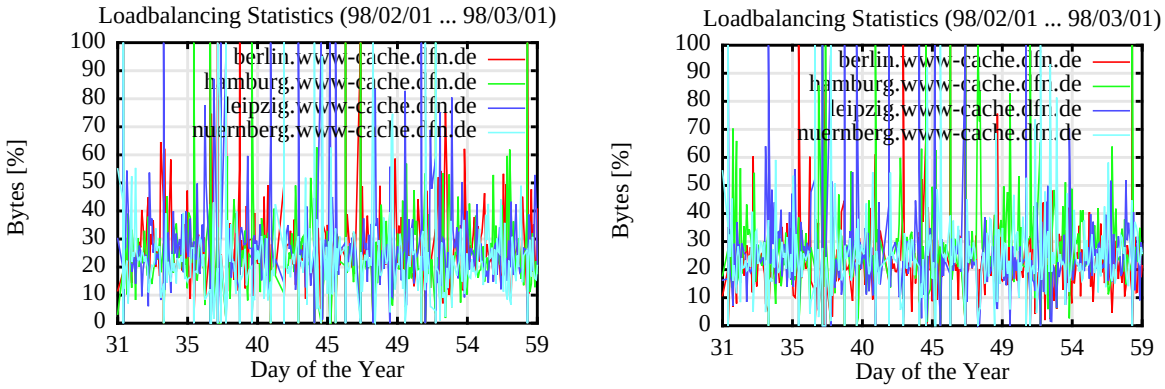

Abbildung A.35: LB (relativ): CARP (left), Squid-CARP (right); total: 8.720.394.922 bytes

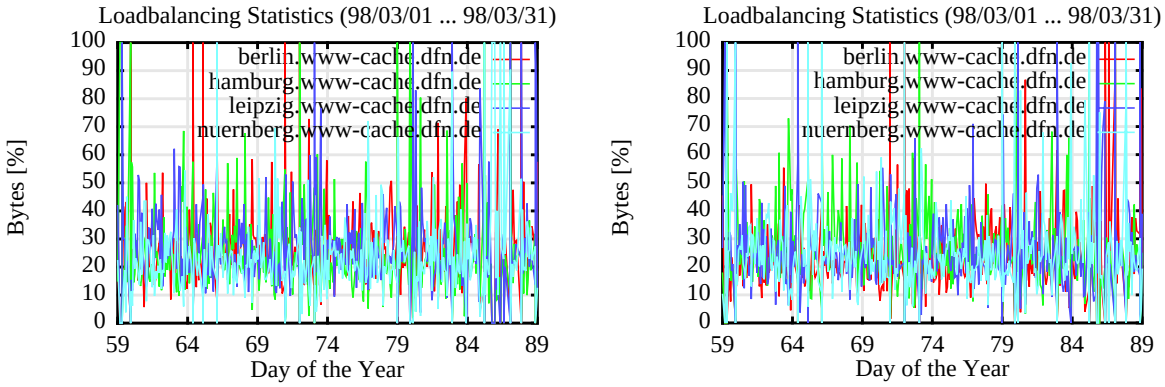

Abbildung A.36: LB (relativ): CARP (left), Squid-CARP (right); total: 10.801.065.979 bytes

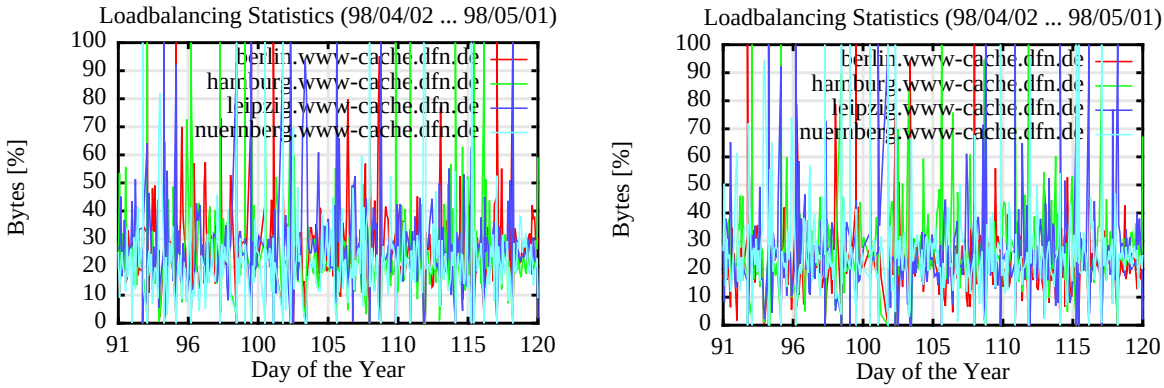

Abbildung A.37: LB (relativ): CARP (left), Squid-CARP (right); total: 12.688.800.455 bytes

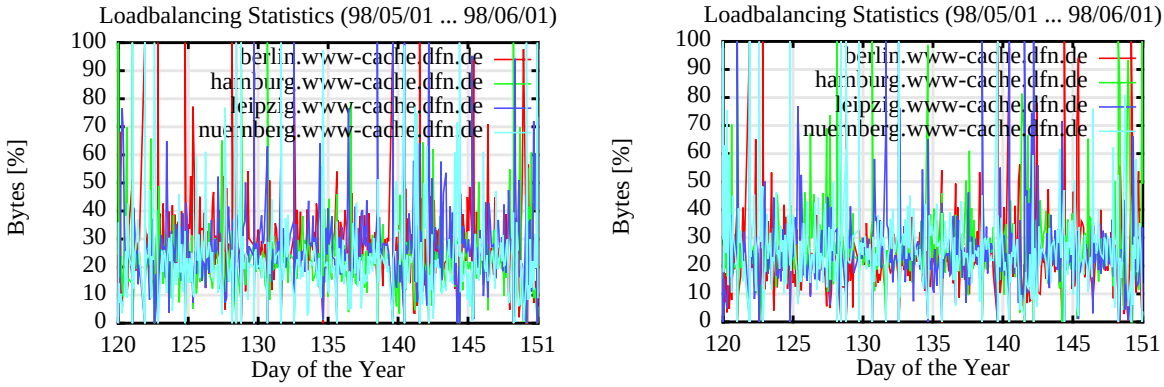

Abbildung A.38: LB (relativ): CARP (left), Squid-CARP (right); total:12.427.657.633 bytes

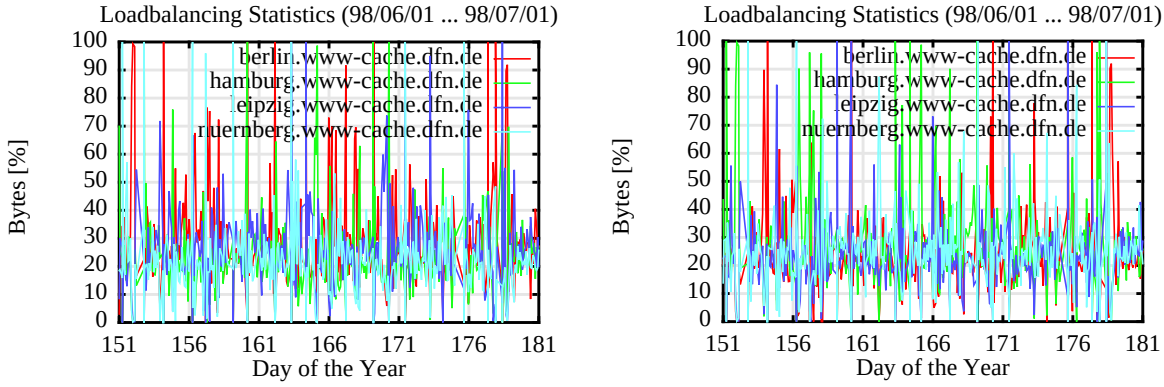

Abbildung A.39: LB (relativ): CARP (left), Squid-CARP (right); total: 11.719.732.474 bytes

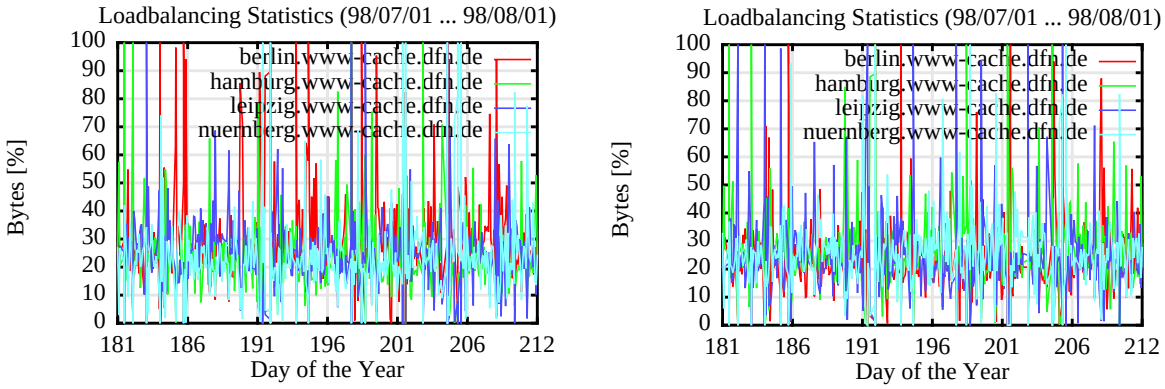

Abbildung A.40: LB (relativ): CARP (left), Squid-CARP (right); total: 11.630.194.766 bytes

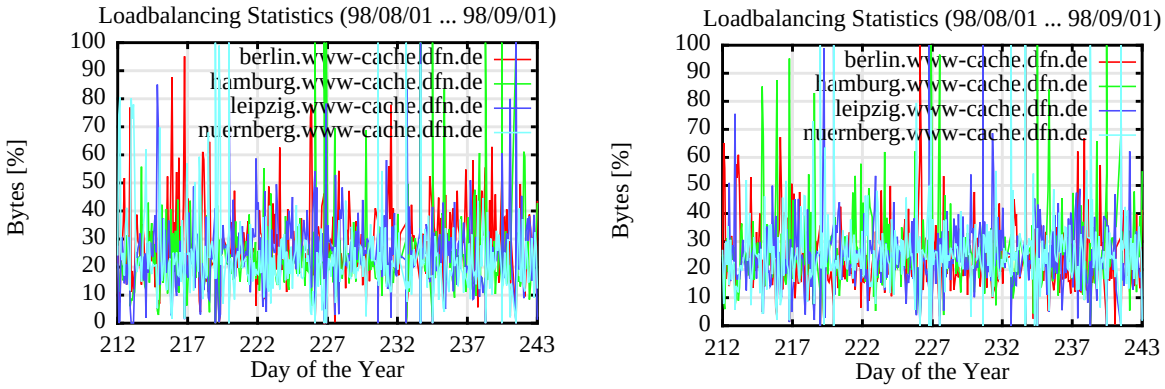

Abbildung A.41: LB (relativ): CARP (left), Squid-CARP (right); total: 14.309.593.596 bytes

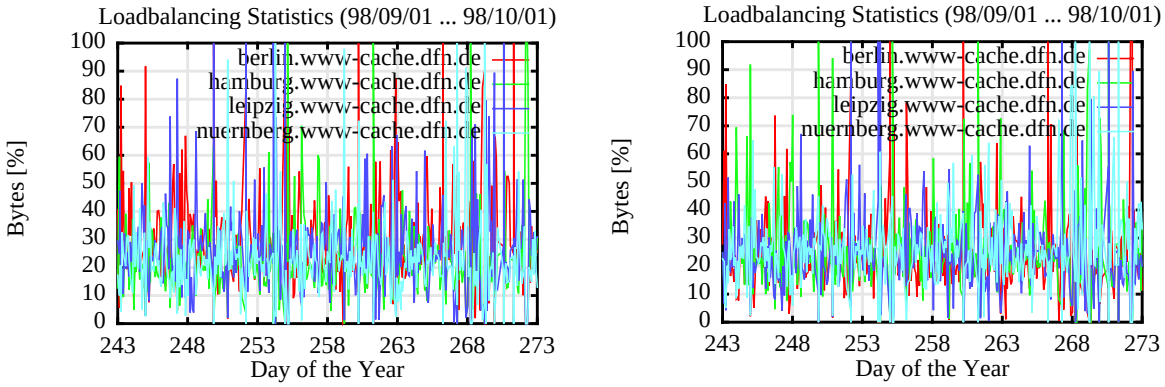

Abbildung A.42: LB (relativ): CARP (left), Squid-CARP (right); total: 11.359.838.505 bytes

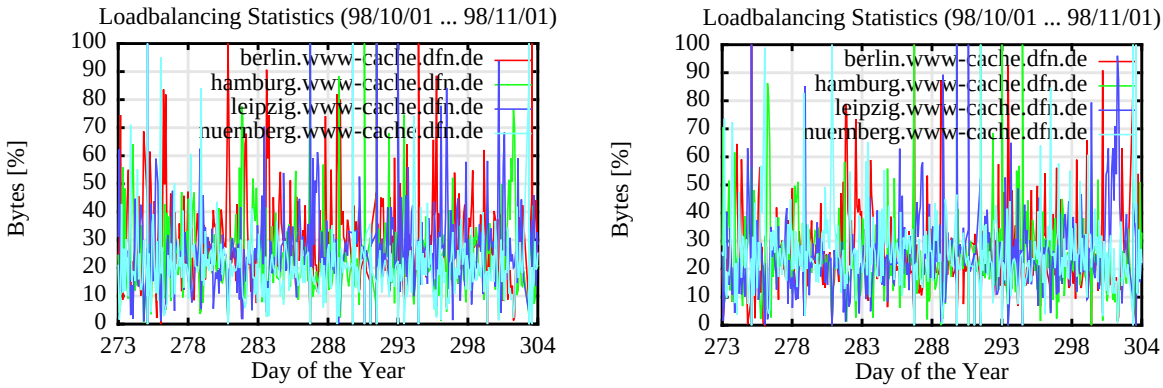

Abbildung A.43: LB (relativ): CARP (left), Squid-CARP (right); total:13.249.527.918 bytes

Anhang B

B.1 Konfiguration des Proxycache.CS.Uni-Magdeburg.De

```
http_port 3128
icp_port 3130

# *** join multicast groups to receive multicasted ICP reques ***
#mcast_groups 239.128.16.128

#        ********************** externe Caches **********************
cache_host leipzig.www-cache.dfn.de sibling 8080 3130 weight=6
cache_host berlin.www-cache.dfn.de sibling 8080 3130 weight=6
cache_host hamburg.www-cache.dfn.de sibling 8080 3130 weight=5
cache_host nuernberg.www-cache.dfn.de sibling 8080 3130 weight=5

#cache_host www-cache.tu-chemnitz.de sibling 8080 3130 weight=5
#cache_host www-cache1.tu-chemnitz.de sibling 8080 3130 weight=5

# *** limit the domains for which a sibling/parent cache will be queried ***
#   --- extern ---
cache_host_domain leipzig.www-cache.dfn.de !.com
cache_host_domain berlin.www-cache.dfn.de .com
cache_host_domain hamburg.www-cache.dfn.de !.com
cache_host_domain nuernberg.www-cache.dfn.de .com

cache_host_domain www-cache.tu-chemnitz.de !.uni-magdeburg.de
#cache_host_domain www-cache1.tu-chemnitz.de !.uni-magdeburg.de

# *** Modify the neighbor type for specific domains ***
# neighbor_type_domain proxycache.cs.uni-magdeburg.de parent .com .edu

# *** list of domains local to your organization ***
#     !!! We do not need this anymore since we are using no_cache ACL !!!
#local_domain uni-magdeburg.de

# *** list of network addresses local to our organization ***
#     !!! Also „obsolete“ because we are using no_cache_acl :) !!!
#local_ip 141.44.0.0

# *** specifies that it is okay to bypass the hierarchy „Pinging“ when there
#     is only a single parent for a given URL ***
#single_parent_bypass off

# *** include the source provider site in squids selection algorithm ***
#source_ping off

# *** This controls how long to wait for replies from neighbor caches ***
#neighbor_timeout 2

# *** not query neighbor caches for certain objects ***
hierarchy_stoplist cgi-bin ? .uni-magdeburg.de/ /mathscinet/
# http://www.brzn.de/cgi-bin http://www.emis.de/cgi-bin

# *** do not cache objects, which include specified patterns ***
cache_stoplist ?
```

```
# ***  do not cache objects, which include specified regex patterns ***
#      ! this might be obsolete by using no_cache ACL tag !
#      (actually for the stop_list_pattern below this is the case ;-))
#cache_stoplist_pattern/i http://.*\.uni-magdeburg\.de/

# *** Use ACL elements to specify uncachable objects. For example,

#     this prevents caching of objects downloaded from servers whose
#     IP addresses are in the specified network ***
#     ! At this point, W3 Server Address must be already in the IP Cache -
#       so this should you give only a very small delay because of the
#       IP cache lookup (guess about 2 ms on a SS20) ... !
acl Local src 141.44.0.0/16
no_cache deny Local

# *** Maximum amout of virtual memory used to cache particularly hot objects ***
cache_mem 256

# *** Maximum about of disk space used by the cache ***
cache_swap 24120

# *** The low- and high-water marks for cache LRU replacement ***
# 2% = 240 MB
# 10% vom System benutzt d.h. effektive Cachegröße = 12 * 0.9 = 10.8 GB
cache_swap_low  94
cache_swap_high 96

# *** The low- and high-water mark for cache memory storage ***
#cache_mem_low 75
#cache_mem_high 90

# *** Objects larger than this size (KB) will NOT be saved on disk ***
maximum_object_size 12228

# *** The size, low-, and high-water marks for the IP cache ***
ipcache_size 2048
#ipcache_low  90
#ipcache_high 95

# *** Directories for on-disk cache storage ***
# jeweils 4 GB
cache_dir /local/cache
cache_dir /local/cache1
cache_dir /local/cache2

# *** Logs the client request activity. Contains an entry for every HTTP
#     and ICP request received.
cache_access_log /local/squid/logs/access.log

# *** Cache logging file (debug messages) ***
cache_log /local/squid/logs/cache.log

# *** Logs the activities of the storage manager ***
cache_store_log /local/squid/logs/store.log

# *** Log User-Agent fields from HTTP requests ***
#useragent_log /local/squid/logs/agent.log

# *** Location for the cache „swap log.“ Normally resides in the first
#     ‘cache_dir’ directory ***
cache_swap_log /local/logs/cache-swap.log

# *** emulate the log file format which many ‘httpd’ programs use ***
```

```
emulate_httpd_log off

# *** Record both the request and the response MIME headers for each HTTP
#     transaction ***
#log_mime_hdrs off

# *** A pathname to write the process-id to ***
pid_filename /var/tmp/squid.pid

# *** Debug Logging options ***
debug_options ALL,1
#20,3 0,1

# *** RFC931/ident lookup of the client username for each connection ***
#ident_lookup off

# *** log fully qualified domain names in the access.log
log_fqdn off

# *** A netmask for client addresses in logfiles and cachemgr output ***
#client_netmask 255.255.255.255

# *** ‘ftpget’ program ***
#ftpget_program /local/squid/bin/ftpget

# *** Options for the ‘ftpget’ program ***
#ftpget_options

# *** anonymous login password ***
#ftp_user squid@

# *** the executable for dnslookup process ***
#cache_dns_program /local/squid/bin/dnsserver

# *** The number of processes spawn to service DNS name lookups ***
dns_children 10

# *** Prevents caches in a hierarchy from interpreting single-component ***
#     hostnames locally.
#dns_defnames off

# *** location of the executable for file deletion process ***
#unlinkd_program /local/squid/bin/unlinkd

# *** location of the executable for the pinger process ***
#pinger_program /local/squid/bin/pinger

# *** location of the executable for the URL redirector ***
redirect_program /local/squid/bin/jesred

# *** The number of redirector processes to spawn ***
redirect_children 10

# *** Relay WAIS request to host (1) at port (2) ***
#wais_relay localhost 8000

# *** Maximum allowed request size in kilobytes ***
request_size 1000

# *** refresh_pattern regex min percent max (minutes), 4320 = 3 days ***
#refresh_pattern        .       0 20% 4320

# *** Selection of TTL’s based on URL regular expressions
```

```
#     Regular expressions are case-sensitive with 'ttl_pattern' and
#     case-insensitive with 'ttl_pattern/i' ***
#     1440 = 1 d,  2880 = 2 d, 10080 = 7 d, 43200 = 30 d
#                                                 Calculate TTL as
#                                    always       this percentage of
#               Regular Expression   fresh if     the object's age
#               matching URLs        age is <=    %age   Max (minutes)
# ---------     --------------------  -----------  ------ ------------
refresh_pattern/i  \.au$                 10080        90%      40320
refresh_pattern/i  \.avi$                10080        90%      40320
refresh_pattern    /cgi-bin/                60        25%      10080
refresh_pattern/i  \.class$              10080        90%      40320
refresh_pattern/i  \.de                   1440        50%      20160
refresh_pattern/i  \.gif$                10080        90%      40320
refresh_pattern/i  \.jpeg$               10080        90%      40320
refresh_pattern/i  \.jpg$                10080        90%      40320
refresh_pattern/i  \.microsoft\.com       1440        75%      40320
refresh_pattern/i  \.midi$               10080        90%      40320
refresh_pattern/i  \.msn\.com   1440        75%      40320
refresh_pattern/i  \.mov$                10080        90%      40320
refresh_pattern/i  \.mpeg$               10080        90%      40320
refresh_pattern/i  \.mpg$                10080        90%      40320
refresh_pattern/i  \.tif$                10080        90%      40320
refresh_pattern/i  \.wav$                10080        90%      40320
refresh_pattern/i  \.xbm$                10080        90%      40320
refresh_pattern    ^http://               1440        50%      40320
refresh_pattern    ^ftp://                2880        50%      40320
refresh_pattern    .                      1440        40%      40320

# *** objects which have not been referenced for this amount of time
#     will be purged from the cache [time unit] ***
#reference_age 12 months

# *** Do NOT continue to retrieve objects from aborted requests ***
#     (min-kbytes percent max-kbytes) ***
#quick_abort    -1 0 0

# *** TTL for failed requests (minutes) ***
#negative_ttl 5

# *** TTL for positive caching of successful DNS lookups (minutes) ***
#positive_dns_ttl 360

# *** TTL for negative caching of failed DNS lookups (minutes) ***
#negative_dns_ttl 5

# *** specifies how long to wait for the connect to complete (seconds) ***
#connect_timeout 120

# *** Abort an active connection after x minutes of no activity on that
#     connection
#read_timeout 15

# *** maximum amount of time that a client (browser) is allowed to
#     remain connected to the cache process (minutes) ***
client_lifetime 60

# *** lifetime to set for all open descriptors during shutdown mode (seconds) ***
shutdown_lifetime 30

# *** How often to force a full garbage collection (minutes) ***
#clean_rate -1
```

```
# *** Access Control Lists ***
# Standard stuff
acl manager proto cache_object
acl Dangerous_ports port 7 9 19
acl SSL_ports port 443 563
acl CONNECT method CONNECT
#acl localhost src 127.0.0.1/32
acl all src 0.0.0.0/0.0.0.0

# Subnetze
acl uni-md src 141.44.0.0/16
#acl uni-md_dom domain .uni-magdeburg.de
#acl sunpool src 141.44.21.0/24
#acl hppool src 141.44.22.0/24
#acl isg src 141.44.23.0/24 141.44.27.0/24
#acl iik src 141.44.24.0/24 141.44.28.0/24
#acl irb src 141.44.18.0/24 141.44.25.0/24
#acl iti src 141.44.26.0/24
#acl atm src 141.44.30.0/24
#acl fin_dom domain .cs.uni-magdeburg.de
#acl no_fin src 141.44.1.0-141.44.17.0/24 141.44.31.0-141.44.254.0/24
acl med-ak src 149.203.0.0/16
acl ifn-md src 192.129.2.0/24
#acl ifn-md_dom domain .ifn-magdeburg.de

acl deny_info url_regex ^http://irb.cs.uni-magdeburg.de/cgi-bin/proxy/denied.*

# Neighbors
# --- intern  proxycache.cs                    141.44.30.31 141.44.20.21
#             proxycache0.urz =
#             infoserv.urz              141.44.251.16
#             proxycache1.cs                   141.44.25.12
#             proxy.med.149.203.102.1
#             siemens.sn.141.44.208.2
acl intern_neighbors src 141.44.30.31/32 141.44.20.31/32 141.44.25.12/32
149.203.102.1/32 141.44.251.16/32 141.44.208.2/32
# --- extern  proxy.zrz.tu-berlin.de           130.149.4.44
#             www-cache.uni-mainz.de           134.93.8.129
#             www-cache.dfn.de                 130.75.15.232
#             columbia.rvs.uni-hannover.de     130.75.5.194
#             www-cache.uni-jena.de            141.35.4.20
#             www-cache.tu-chemnitz.de         134.109.132.36
#             www-cache1.tu-chemnitz.de        134.109.132.30
#             proxy.uni-leipzig.de             139.18.25.3
#             hh-cache.dkrz.de136.172.119.33
#             www-cache.ruhr-uni-bochum.de     134.147.222.9
#             mkdir.boerde.de                  193.175.28.19
#             relay.boerde.de                  193.175.28.66  (backup)
acl extern_neighbors src 130.149.4.44/32 134.93.8.129/32 130.75.15.232/32
130.75.5.194/32 141.35.4.20/32 134.109.132.36/32 134.109.132.30/32 139.18.25.3/32
136.172.119.33/32 134.147.222.9/32 193.175.28.19/32 193.175.28.66/32

# Wer nutzt mich:
# Proxycache0-4 + MedAk Uni-Magdeburg, Siemens.sn
# TU Berlin, FSU Jena, eV relay|mkdir.boerde.de
#
# Wer nutzt mich nicht mehr:
# Uni Bochum, Uni Mainz, Uni Hannover, TU Leipzig, DKRZ Hamburg, TU Chemnitz

# Management Hosts
acl herkules src 141.44.25.22/32

http_access deny Dangerous_ports
```

```
http_access allow deny_info
http_access allow uni-md
http_access allow med-ak
#http_access allow ifn-md
# intern_neighbors werden durch uni-md impliziert!
#   http_access allow intern_neighbors
# gleiches gilt für herkules
#   http_access allow herkules
http_access allow extern_neighbors
http_access allow CONNECT SSL_ports
http_access deny manager !herkules
http_access deny all

# *** return a HTTP redirect for requests which do not pass the
#     'http_access' rules ***
#deny_info URL acl
deny_info http://ivs.cs.uni-magdeburg.de/cgi-bin/proxy/denied-proxycache.cs.uni-
magdeburg.de.cgi all

icp_access allow intern_neighbors
icp_access allow extern_neighbors
icp_access deny all

#miss_access allow intern_neighbors
miss_access allow all

# *** like 'cache_host_domain' but provides more flexibility by using ACL's ***
# cache_host_acl cache-host       [!]aclname ...

# *** Email-address of local cache manager  ***
cache_mgr cachemgr@cs.uni-magdeburg.de

# *** EUID/EGID of the running processes ***
cache_effective_user nobody nobody

# *** alternative hostname of the machine running the cache (alias) ***
visible_hostname proxycache.CS.Uni-Magdeburg.De

# *** announcing our cache every n hours (168 h = 7 d) ***
cache_announce 168

# *** hostname and portnumber where the registration message will be sent ***
#announce_to tracker.ircache.net:3131

# *** The DNS tests exit as soon as the first site is successfully looked up ***
dns_testnames internic.net usc.edu cs.colorado.edu mit.edu yale.edu

# *** number of logfile rotations to make upon receiving a USR1 signal ***
logfile_rotate 10

# *** Appends local domain name to hostnames without any dots in them ***
append_domain .cs.uni-magdeburg.de

# *** Size of receive buffer to set for TCP sockets ***
#tcp_recv_bufsize 0

# *** hostname and port number where all SSL requests should be forwarded to ***
#ssl_proxy cache_host
#ssl_proxy host:port
#ssl_proxy none

# *** name of a 'cache_host' or a hostname and port number where all non-GET
#     (i.e. POST, PUT) requests should be forwarded to
```

```
#passthrough_proxy none

# *** Proxy authentication ***
#proxy_auth /dev/null

# *** HTML text to include in error messages ***
err_html_text <HR>Falls Sie der Meinung sind, da&szlig; dieser Fehler auf eine
Fehlfunktion des Proxy Caches zur&uuml;ckzuf&uuml;hren ist, benachrichtigen Sie
bitte umgehend die <i><b><a href="mailto:cachemgr@cs.uni-
magdeburg.de">&lt;cachemgr@cs.uni-magdeburg.de&gt;</a></b></i>. Vergessen Sie
nicht, den Fehler Code, die abgefragte URL, die Version ihres WWW-Browsers sowie
andere evtl. relevanten Informationen beizuf&uuml;gen.

# *** request UDP_HIT_OBJ replies from neighbors ***
udp_hit_obj on

# *** Limit UDP_HIT_OBJ size to be less than this value. Set to zero to ***
#     select the size permited by the socket
#udp_hit_obj_size     0

# *** keep pools of allocated (but unused) memory available for future use ***
#memory_pools on

# *** include my system's IP address or name in the HTTP requests it
#     forwards ***
forwarded_for on

# *** By default, ICP queries are logged to access.log ***
#log_icp_queries on

# *** direct fetches for sites which are no more than this many hops away ***
minimum_direct_hops 4

# *** Specify passwords for cachemgr operations ***
# shutdown *            # log/status *          # log/enable *
# log/disable *         # log/clear *           # log *
# info                  # stats/objects         # stats/vm_objects
# stats/utilization     # stats/ipcache         # stats/fqdncache
# stats/dns             # stats/redirector      # stats/io
# stats/reply_headers   # stats/filedescriptors # stats/netdb
# parameter             # server_list           # client_list
# squid.conf *
#
cachemgr_passwd foobar squid.conf stats/vm_objects stats/objects

# *** Number of first-level directories to create for storing ***
#     cached objects.  Minimum 1, maximum 256, default 16
#swap_level1_dirs 16

# *** Number of sub-directories to create under each first-level ***
#     directory.  Minimum 1, maximum 256, default 256
#swap_level2_dirs 256

# *** Average object size, used to estimate number of objects your
#     cache can hold ***
# default 13
store_avg_object_size 20

# *** Target number of objects per bucket in the store hash table ***
store_objects_per_bucket 20

# *** Filter out certain HTTP request headers for privacy reasons ***
#     {off | standard | paranoid }
```

```
http_anonymizer off

# *** strip User-agent string from the request ***
#fake_user_agent none

# *** collecting per-client statistics ***
#client_db on

# *** The low and high water marks for the ICMP measurement database ***
#netdb_low 900
#netdb_high 1000

# *** The minimum period for measuring a site ***
#netdb_ping_period 10 minutes

# *** ask peers to include ICMP data in their ICP replies ***
#query_icmp off

# *** return ICP_HIT for stale cache objects ***
#icp_hit_stale off

# *** minimum_retry_timeout - specifies the minimum connect timeout for the
#     retry patch, for instances when the connect timeout is reduced to
#     to compensate for the availability of multiple IP addresses ***
#minimum_retry_timeout 5

# *** Set the maximum number of connection attempts for a host that only has
#     one address (for multiple-address hosts, each address is tried once)
#     for the retry patch. ***
#maximum_single_addr_tries 3

# *** turn ‘Pragma: no-cache’ requests into If-Modified-Since requests ***
#reload_into_ims off
```

B.2 Squid Redirector Konfiguration (Jesred 1.2)

B.2.1 Jesred Konfigurationsdatei

```
# file with IP addresses, for which URL rewriting is [not] allowed
allow = /local/squid/etc/redirect.acl

# file with rules for URL rewriting
rules = /local/squid/etc/redirect.rules

# log file for general, error and debug messages (empty value or commenting
# this out disables logging)
redirect_log = /local/squid/logs/redirect.log

# log file for URL rewrites (empty value or commenting this out disables
# logging of URL rewrites)
rewrite_log = /local/squid/logs/rewrite.log

# Debug mode: if set to yes and DEBUG option was compiled in, this enables
# debug logging to redirect_log
#debug = true

# Allow ICP_QUERY request to be rewritten, if a rule applies
siblings = true
```

B.2.2 Jesred ACL Konfigurationsdatei

```
# Example: IP access pattern file

# Jesred: http://ivs.cs.uni-magdeburg.de/~elkner/webtools/jesred/

# the following CDIR notation is allowed ONLY:
# a.b.c.d/mwith a, b, c, d ... {0..255} andm ... {0..32}
# and is referred as IP access pattern.
#
# you can prefix any IP access pattern with a '!`, which explicitly says,
# „Do not rewrite the passed URL from this client“

##############################################################################
# Since jesred uses a linear list of ip access patterns, the order of the   #
# ip access patterns in this file is important!!!                            #
##############################################################################

# jesred uses a linear IP Access pattern list. As long as no match is found,
# it compares the clients IP address with the next pattern, listed below.
# If the end of the list is reached and no match was found, jesred echo's
# back a „\n“, which indicates no URL replacement to squid.
# If the client IP address matches an entry below, rewrite rules
# are applied immediately (i.e. no further checks for other IP access
# pattern matches will be done).
#

# These are my children caches which have their own redirectors running
#!141.44.251.15/32
#!149.203.102.1/32
# rewrite all URLs from
141.44.0.0/16
149.203.0.0/16
193.175.28.0/24
```

B.2.3 Jesred Redirect Rules Konfigurationsdatei

```
# Example: redirector rules

# Jesred: http://ivs.cs.uni-magdeburg.de/~elkner/webtools/jesred/

# since jesred uses exactly the same pattern functions as squirm 1.0 betaB,
# you can also find detailed information about this file on:
# http://www.senet.com.au/squirm/#squirm_patterns

##############################################################################
# Since jesred uses a linear list of redirect rules, the order of the rules #
# in this file is important!!!                                              #
##############################################################################

# TAG: abort
#
# If jesred enconters the specified extension
# in the passed URL, it immediately returns and echo's back a newline (i.e.
# no rewrite) - so this speeds up the lookup process a lot!

# TAG: regex RE RURL
# TAG: regexi RE RURL
#
# regex  ... indicates, that the following RE is case-sensitive
# regexi ... indicates, that the following RE is case-insensitive
# RE     ... is the regular expression, which has to match the passed URL to get
#            rewritten with the following RURL (it is only limited by the
```

```
#         implementation of the used regex functions - so see the man page
#            for the regex functions you compiled jesred with, to get detailed
#            information on supported RE's)
# RURL   ... if RE matches the passed URL, jesred returns RURL

# I recommend to use the „accelerators“ '^' and '$' in REs whereever it is
# possible, since this speeds up the pattern matching a lot! For more
# information see: http://www.senet.com.au/squirm/#squirm_patterns

# javatalk.excite.com
# 1 - 2
regex.*\/ad.htmlhttp://141.44.30.2/proxy/empty.html
regex^http://www.ad-server.de/.*(ard[0-9]*|indexframe).*.html$ http://
141.44.30.2/proxy/empty.html

# 3 - 9
abort.html
abort.jpg
abort.html
abort.shtml
abort.java
abort.jar
abort.htm

# 10
regex^http://199.78.52.10/~web_ani/.*\.gifhttp://141.44.30.2/images/dot.gif
# 11 www.ad-server.de(www.tvspielfilm.de)
regex^http://.*/ad_gifs/.* http://141.44.30.2/images/dot.gif

# ads.web.de    (www.web.de)
# 12
regexi^http://ads.web.de/.*gifhttp://141.44.30.2/images/dot.gif

# 13
abort.gif

# 14 - 15
regexi^http://ad\..*doubleclick.net/(ad|viewad)/.*http://141.44.30.2/images/
dot.gif
regex^http://ads[0-9][0-9].focalink.com/SmartBanner/nph-graphic.*http://
141.44.30.2/images/dot.gif

# 209.67.30.71
# ads.zdnet.com
# 16
regexi^http://.*/accipiter/ads.*http://141.44.30.2/images/dot.gif

# 17 - 21
regex^http://tracker.clicktrade.com/Tracker.*http://141.44.30.2/images/dot.gif
regex^http://adforce.imgis.com/?adserv.*http://141.44.30.2/images/dot.gif
regex^http://195.90.252.40/banner.*http://141.44.30.2/images/dot.gif
regex^http://www.artuframe.com/partners/affiliates/banners.*http://141.44.30.2/
images/dot.gif
regex^http://ad.adsmart.net/ads/.*http://141.44.30.2/images/dot.gif

# advert.heise.de(www.heise.de)
# 22
regex^http://advert.heise.de/.*oma.(pict|count)/.* http://141.44.30.2/images/
dot.gif

# 23 - 28
regex^http://adservant.mediapoint.de/cgi-object/webdriver/.* http://141.44.30.2/
images/dot.gif
```

```
regex^http://(www.){0,1}kaufwas.com/cgi-bin/.*type=gfx.* http://141.44.30.2/
images/dot.gif
regex^http://www.e-lite.net/cgi-bin/view.pl.* http://141.44.30.2/images/dot.gif
regex^http://.*.linkexchange.com/.*/logoshowad.* http://141.44.30.2/images/
dot.gif
regex^http://.*fpcount.exe/hitmaker/.* http://141.44.30.2/images/dot.gif
regexi^http://.*\/(AdSwap|ad|banner|barimage|bserve|count|counter|go/
redirect|hitbox|link|show|showbanner|showbar|trackrun|top|zeige)\.{0,1}(dll|exe|p
l|cgi|gifx*){0,1}\?.* http://141.44.30.2/images/dot.gif

# w[0-9]+.hitbox.com (www.pennyweb.com)
# 29
regexi^http://.*\.hitbox\..*/w./.* http://141.44.30.2/images/dot.gif

# ad.asv.de
# www.cityseaysearch7.com
# adserv.spiegel.de
# 30
regex^http://.*/RealMedia/ads/adstream_.* http://141.44.30.2/images/dot.gif

# adserv.spiegel.de(www.spiegel.de)
# 31
regex^http://.*/cgi-bin/vdz/AD/homepage$ http://141.44.30.2/images/dot.gif

# ad.preferences.com
# ads.csi.com
# 32 - 33
regex^http://.*/image\.ng.* http://141.44.30.2/images/dot.gif
regex^http://.*/adverts/imp/.* http://141.44.30.2/images/dot.gif

# adex3.flycast.com(www.nettaxi.com)
# 34
regex^http://adex3.flycast.com/server/socket.* http://141.44.30.2/images/dot.gif

# banners.pennyweb.com(www.pennyweb.com)
# 35
regex^http://banners.pennyweb.com/.* http://141.44.30.2/images/dot.gif

# www.adz.net   (www.pennyweb.com)
# 36
regexi^http://www.adz.net/goto/.* http://141.44.30.2/images/dot.gif

# www.clickers.net(www.pennyweb.com)
# 37
regexi^http://www.clickers.net/Banners/.* http://141.44.30.2/images/dot.gif

# www.geocities.com(www.geocities.com)
# 38
regexi^http://www.geocities.com/ad_container/.* http://141.44.30.2/proxy/
empty.html

# search.excite.com(www.excite.com)
# 39
regexi.*/img/ads/.*http://141.44.30.2/images/dot.gif

# www.sys-con.com(www.sys-con.com)
# 40
regexi.*/ad_banners/.*http://141.44.30.2/images/dot.gif

# NOTE: actually '.' in RE is any character, so if you want to be sure,
#       escape the special meaning with a prefixed '\' ;-)

# We use the IP address in the rewritten URL to get the local image cached ;-)
```

B.3 PCMS-Konfigurations-Datei für die MCH

```
# GPS4A configuration file

# mrtg: the mrtg executable
mrtg = /localapp/mrtg-2.5.2/mrtg

# icondir: path from httpd servers DocumentRoot where the mrtg icons
# ( mrtg-l.gif, mrtg-m.gif, mrtg-r.gif, mrtg-ti.gif ) are located
# default = /icons/mrtg/
#           which might be on your fs /local/www/htdocs/icons/mrtg/
icondir = /icons/mrtg

# datadir: path where to archive the obtained data from each run of GPS4A
# !!! only required, if you want to archive the obtained data in
#     appropriate files !!!
datadir=/local/reserve1/proxystats

# htmldir: system path, where the html files and gifs will be stored
htmldir = /home/irb/elkner/public_html/stats/proxy

# testurl: The URL, which is used by echoping to make measurements about
# cache reponse times
testurl = http://141.44.30.2/test/eping.html

# echoping: where do we find the program echoping
echoping = /usr/local/bin/echoping

# now the proxy caches to monitor in the following format
# KEYWORD ADDRESS PORT PASSWORD
#
# NOTE: 1) if no password is required for obtaining the cache objects
#          „info“, „stats/dns“ and „stats/utilization“ use the password ‘no’
#       2) the keywords must be unique and are used to store the html pages
#          and gifs in $htmldir/$keyword
#       3) more than 1 whitespace are ignored

cache ProxyCache.CS.Uni-Magdeburg.De3128no
cache0WWW-Cache.Uni-Magdeburg.DE3128foobar
#cache1ProxyCache1.CS.Uni-Magdeburg.De3128no
#cache2ProxyCache2.CS.Uni-Magdeburg.De3128no
#cache3ProxyCache3.Math.Uni-Magdeburg.De 3128no
cache4Proxy.Med.Uni-Magdeburg.De3128no
```

Literaturverzeichnis

[1] Charlie Kindel, Lou Montulli, Eric Sink, Wayne Gramlich, Jonathan Hirschman, Tim Berners-Lee,Dan Connolly, Bruce Kahn, Steve Byrne. Inserting objects into HTML - W3C Working Draft, 18. Februar 1997.
http://www.w3.org/pub/WWW/TR/WD-object-970218

[2] F. Anklesaria, M. McCahill, P. Lindner, D. Johnson, D. Torrey, B. Alberti. Network Working Group RFC: 1436 - The Internet Gopher Protocol, März 1993.
ftp://ftp.internic.net/rfc/rfc1436.txt

[3] Network Wizards. Internet Domain Survey, Juli 1991 - Januar 1997.
http://www.nw.com/zone/summary-reports/

[4] Mark K. Lottor. Network Working Group RFC: 1296 - Internet Growth (1981-1991), Januar 1992.
http://www.nw.com/zone/rfc1296.txt

[5] Merit Network, Inc, Graphics, Visualization and Usability Center of Georgia Institute of Technology. GVU's NSFNET Backbone Statistics. Dezember 1992 - April 1995.
http://www.cc.gatech.edu/gvu/stats/NSF/

[6] Syed S. Towheed. NASA's Use of the World Wide Web to Deliver Shoemaker-Levy 9 Collision Data in Near-Real Time. Zuletzt geändert: 1. April 1996.
http://nssdc.gsfc.nasa.gov/misc/www_conf/towheed.html

[7] O. Beyer; H. Hackel; V. Pieper; J. Tiedge. Wahrscheinlichkeitsrechnung und mathematische Statistik. Teubner Verlag Stuttgart, Leipzig. 7. neubearb. Auflage, 1995.

[8] T. Berners-Lee, L. Masinter, M. McCahill. RFC: 1738 - Uniform Resource Locators (URL), Dezember 1994.
ftp://ds.internic.net/rfc/rfc1738.txt

[9] Fielding, et. al. RFC 2068 - Hypertext Transfer Protocol -- HTTP/1.1, Januar 1997.
http://www.w3.org/pub/WWW/Protocols/rfc2068/rfc2068

[10] Berners-Lee, et. al. RFC 1945 - Hypertext Transfer Protocol -- HTTP/1.0, May 1996.
http://www.w3.org/pub/WWW/Protocols/rfc1945/rfc1945

[11] W. Dörfler; W. Peschek. Einführung in die Mathematik für Informatiker. Hanser Verlag, München; Wien. Völlig neu bearb. Ausg. d. 2bd. Werkes “Dörfler, Mathematik für Informatiker”, 1988, S 203f..

[12] D. Wessels; K. Claffy. RFC 2186 - Internet Cache Protocol (ICP), version 2. National Laboratory for Applied Network Research / UCSD, September 1997.
http: / / squid.nlanr.net / Squid / rfc2186.txt

[13] D. Wessels; K. Claffy. RFC 2187 - Application of Internet Cache Protocol (ICP), version 2. National Laboratory for Applied Network Research / UCSD, September 1997.
http: / / squid.nlanr.net / Squid / rfc2187.txt

[14] M. Bowman; P. Danzig; D. Hardy; Manber U.; Schwartz M; D. Wessels. The Harvest Information Discovery and Access System. Internet Research Task Force - Resource Discovery.
http: / / harvest.transarc.com

[15] D. Wessels. The Squid Internet Object Cache. National Laboratory for Applied Network Research.
http: / / squid.nlanr.net

[16] P. Danzig. NetCache Architecture and Deployment. Network Appliance, Inc.
http: / / www.netapp.com / products / internet.html

[17] White Paper: Shared Network Caching and Cisco´s Cache Engine. Cisco Systems, Inc., 29. April 1998
http: / / www.cisco.com / warp / public / 751 / cache / cds_wp.htm

[18] Cisco Cache Engine Frequently Asked Questions. Cisco Systems, Inc., 13. April 1998
http: / / www.cisco.com / warp / public / 458 / 36.html

[19] V. Valloppillil, K. W. Ross. INTERNET-DRAFT <draft-vinod-carp-v1-03.txt>: Cache Array Routing Protocol v1.0. Microsoft Corporation,University of Pennsylvania, 26. Februar 1998.
http: / / www.eu.microsoft.com / proxy / guide / CarpSpec.asp

[20] P. Vixie. INTERNET-DRAFT <draft-vixie-htcp-proto-00.txt>: Hyper Text Caching Protocol (HTCP / 0.0), Vixie Enterprises, März 1998.
ftp: / / ftp.isi.edu / internet-drafts / draft-vixie-htcp-proto-00.txt

[21] A. Rousskov. E-Mail: Re: HTCP versus ICP, 30. Juni 1998.
http: / / squid.nlanr.net / Mail-Archive / squid-users / 9806 / archive / 0527.html

[22] A. Rousskov, D. Wessels. Cache Digests, National Laboratory for Applied Network Research, 17. April 1998.
http://wwwcache.ja.net/events/workshop/31/rousskov@nlanr.net.ps

[23] R. Rivest: RFC 1321 - The MD5 Message-Digest Algorithm. MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992.
ftp://www.cs.tu-bs.de/pub/docs/internet/rfc/rfc-1300-1399/rfc1321.txt.gz

[24] B. Bloom, Space/time trade-offs in hash coding with allowable errors, Communications of the ACM, Vol. 13, S. 422-426, Juli 1970.

[25] A. O. Freier, P. Karlton, P. C. Kocher. The SSL Protocol Version3.0, Netscape Communications, 18. November 1996.
http://home.netscape.com/eng/ssl3/draft302.txt

[26] D. Wessels. SQUID Frequently Asked Questions, National Laboratory for Applied Network Research, 1998.
http://squid.nlanr.net/Squid/FAQ/FAQ-1.html

[27] H. Pralle, et. al. Konzeption einer Cache-Server-Infrastruktur auf dem Wissenschaftsnetz, Verein zur Förderung eines Deutschen Forschungsnetzes e.V. (DFN-Verein), Hannover, August 1996.
http://www-cache.dfn.de/Cache/Project.html

[28] J. Mauro. Inside Solaris: Fiddling around with files, part one, Sun Microsystems, Februar 1998.
http://www.sunworld.com/sunworldonline/swol-03-1998/swol-02-insidesolaris.html

[29] J. Mauro. Inside Solaris: Fiddling around with files, part two, Sun Microsystems, März 1998.
http://www.sunworld.com/sunworldonline/swol-03-1998/swol-03-insidesolaris.html

[30] Jens Elkner. Jesred - a redirector for squid. Institut für Verteilte Systeme, Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg, Magdeburg, August 1998.
http://ivs.cs.uni-magdeburg.de/~elkner/webtools/jesred/

[31] Jens-S. Vöckler. Solaris 2.x - tuning your TCP/IP stack and more. Institut für Rechnernetze und Verteilte Systeme, Universität Hannover, September 1998.
http://www.rvs.uni-hannover.de/people/voeckler/tune/EN/tune.html

[32] Adrian Cockroft. Sun Performance and Tuning - Java and the Internet. 2nd Edition, SUN Microsystems Inc., April 1998

[33] V. Jacobsen, R. Braden, D. Borman. RFC 1323: TCP Extensions for High Performance. University of California - Lawrence Berkeley Laboratory, University of Southern California - Information Science Institute, Cray Research, Mai 1992.
ftp://nic.merit.edu/internet/documents/rfc/rfc1323.txt

[34] W. R. Stevens. TCP/IP Illustrated, Volume 1 - The Protocols. Addison-Wesley Publishing Company, Reading, MA, 1994.
http://www.kohala.com/~rstevens/tcpipiv1.html

[35] W. R. Stevens. TCP/IP Illustrated, Volume 2 - The Implementation. Addison-Wesley Publishing Company, Reading, MA, 1995.
http://www.kohala.com/~rstevens/tcpipiv2.html

[36] W. R. Stevens. TCP/IP Illustrated, Volume 3 - T/TCP, HTTP, NNTP, Unix Domain Sockets. Addison-Wesley Publishing Company, Reading, MA, 1994.
http://www.kohala.com/~rstevens/tcpipiv3.html

[37] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, E. A. Fox. Caching Proxies: Limitations and Potentials. Technical Report TR 95-12. Department of Computer Science, Virginia Polytechnic Institute and State University, 17. Juli 1995.
http://www.cs.vt.edu/~chitra/docs/95www4ASAWF/WWW4.ps.gz

[38] A. Bestavros, R. L. Carter, M. E. Crovella, C. R. Cunha, A. Heddaya, S. A. Mirdad. Application-Level Document Caching in the Internet. in IEEE SDNE'96: The Second International Workshop on Services in Distributed and Networked Environments, Whistler, British Columbia, June 1995.
http://www.cs.bu.edu/~best/res/papers/sdne95.ps

[39] S. Glassman. A Caching Relay for the World Wide Web. In First International World-Wide Web Conference. Elsevier Science, Genf, Mai 1994, S. 69 - 76.
http://www1.cern.ch/PapersWWW94/steveg.ps

[40] l. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker. On the Implications of Zipf's Law for Web Caching. Xerox Palo Alto Research Center / University of Wisconsin, Madison / University of Southern California, Los Angeles.
http://www.cs.wisc.edu/~cao/papers/zipf-implication/index.html

[41] D. Raggett et al. HTML 3.2 Reference Specification - W3C Recommendation. World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University), 14. Januar 1997
http://www.w3.org/TR/REC-html32.html

[42] D. Raggett, A. L. Hors, I. Jacobs et al.HTML 4.0 Specification - W3C Recommendation. World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University), überarbeitet am 24. April 1998.
http://www.w3.org/TR/REC-html40/

[43] A. Powell. Resolving DOI Based URNs Using Squid - An Experimental System at UKOLN. University of Bath. D-Lib Magazine. Juni 1998.
http://www.dlib.org/dlib/june98/06powell.html

[44] D. Kristol,L. Montulli. RFC 2109: HTTP State Management Mechanism. Bell Laboratories, Lucent Technologies und Netscape Communications Corporation, Februar 1997.
ftp://nic.merit.edu/internet/documents/rfc/rfc2109.txt

[45] Netscape Support Documentation: Navigator Proxy Auto-Config File Format. Netscape Communications Corporation, März 1996.
http://home.netscape.com/eng/mozilla/2.0/relnotes/demo/proxy-live.html

[46] JavaScript Documentation.Netscape Communications Corporation, 17. November 1998.
http://developer.netscape.com/docs/manuals/javascript.html

[47] White Paper: Super Proxy Script - How to make distributed proxy servers by URL hashing. Sharp Corporation, August 1996-1998.
http://naragw.sharp.co.jp/sps/index.html

[48] K. Egevang, P. Francis. RFC 1631: The IP Network Address Translator (NAT). Cray Communications und NTT Software Lab, Mai 1994.
ftp://nic.merit.edu/internet/documents/rfc/rfc1631.txt

[49] J. Elkner. Squid 1.1 Information and Documentation. Institut für Verteilte Systeme. Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg, Oktober 1995 - 1998.
http://ivs.cs.uni-magdeburg.de/~elkner/proxy/Squid/

Selbständigkeitserklärung

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

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Jens Elkner

Magdeburg, den 01. Februar 1999

Thesen

- Der durch das WWW verursachte Verkehr benötigt mehr Bandbreite, als irgendein anderer Dienst im Internet. Vorhandene Extranets sind in der Regel diesem Strom an Informationen nicht gewachsen, noch stehen genügend finanzielle Mittel und Möglichkeiten zum Ausbau der vorhandenen Strukturen zur Verfügung. Ebenso wird die Nutzung des Extranets oftmals volumenabhängig in Rechnung gestellt. Deshalb ist Proxycaching ein Weg zur **Reduzierung bzw. Begrenzung der Kosten für die Nutzung von Internetzugängen**.
- Aufgrund der nur begrenzt zur Verfügung stehenden Bandbreite im Extranet und daraus resultierenden Netzüberlastungen, sowie Überlastung von WWW-Servern durch sehr viele Zugriffe, ist das Herunterladen von WWW-Objekten für den Endnutzer oftmals eine sehr langwierige Sache. Deshalb bieten Proxycaches im Intranet oder im schnellen Extranet (z.B. B-WiN) eine Möglichkeit, eine **Reduzierung von Verzögerungen im WWW** zu erreichen.
- Auch wenn die zu zahlenden Summen für die Nutzung eines Internetzugangs immer geringer werden, Bandbreite wird immer etwas kosten. Deshalb ist ein Proxycache immer zumindest ein kleiner Beitrag, einen sogenannten **Return of Investment** zu erzielen.
- Aufgrund unterschiedlicher finanzieller und umweltbedingter Verhältnisse werden nie alle Internet-Teilnehmer (ob Client oder Server) über die gleiche Bandbreite verfügen. Deshalb können Proxycaches auch helfen, einen **Ausgleich von Unterschieden bzgl. Bandbreite und daraus entstehende Verzögerungen** zu schaffen.
- Immer mehr Menschen machen sich das WWW zu nutze, sei es privat oder geschäftlich. Die Teilnehmerzahl im Internet steigt ständig und auch multimediale Angebote inklusive großen Sound- und Animationsdateien sowie Bilder sind immer häufiger zu finden. Die Folge ist: Der **Bedarf an Bandbreite wird weiterhin steigen**.
- Sogenannte **hot spots wird es immer im WWW geben**. Hot spots sind WWW-Server, auf den sich besonders viele Anfragen für eine bestimmte Zeit beziehen, wie z.B. den WWW-Server für die Olympischen Winter-Spiele in Japan oder NASA-Weltraummissionen. Werden nicht genügend Vorkehrungen getroffen, sind diese Server oder die Leitungen zu diesen servern hoffnungslos überlastet. Proxycaches sind eine Möglichkeit, dies zu verhindern.
- **Rechnerzubehör wird immer preiswerter**. Deshalb kann der Kauf von Rechnerzubehör wie z.B. RAM oder Festplatten oder sogar eines neuen Rechners für einen Proxycache bedeutend günstiger sein, als der zu zahlende Preis für die Bereitstellung der Bandbreite mit äquivalente Leistungen.