

Konzeption und praktischer Einsatz eines Proxycache-Systems zur Erhöhung der Dienstgüte im WWW

Dipl.-Inf. Jens Elkner, Prof. Dr.-Ing. habil. Reiner Dumke, Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik

Kurzfassung

Die Dienstgüte im World Wide Web (WWW) wird vor allem durch die Durchsatzrate und die dadurch implizierte Zugriffsgeschwindigkeit geprägt. Dabei können spezielle Speicherungsformen, wie zum Beispiel die Proxycache-Technik, wesentlich zur Verbesserung dieser Dienstgüte beitragen. Allerdings sind dabei die unterschiedlichsten Einflußfaktoren, wie die verteilt oder lokal verfügbare Speichergröße oder die Speicherdauer, innerhalb eines sich dynamisch verhaltenen Internets zu beachten. Der vorliegende Beitrag stellt eine Methodik zur effizienten WWW-Nutzung auf der Grundlage eines Proxycache-Verbundes vor. Ausgangspunkt sind umfangreiche Analysen des dynamischen Verhaltens von Web-Applikationen und die dadurch mögliche effiziente Speichereinrichtung. Für eine kontinuierliche Analyse und deren Anwendung zur Effizienzerhaltung wurden für einen Squid-basierten Proxycache-Einsatz Tools implementiert, die unter <http://ivs.cs.uni-magdeburg.de/~elkner/webtools/> einzusehen bzw. zu nutzen sind.

1 Einleitende Bemerkungen

Der exponentielle Anstieg von WWW-Servern und der darauf enthaltenen Informationen/Dienste bewirkt auch ein starkes Ansteigen des Verkehrs im Internet. So zeigt Abbildung 1 den Zuwachs an transferierten Bytes bzgl. verschiedener Protokolle im NSFNET¹ [3]. Eine genauere Analyse der Daten 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 WWW² jedoch um ca. 25%!

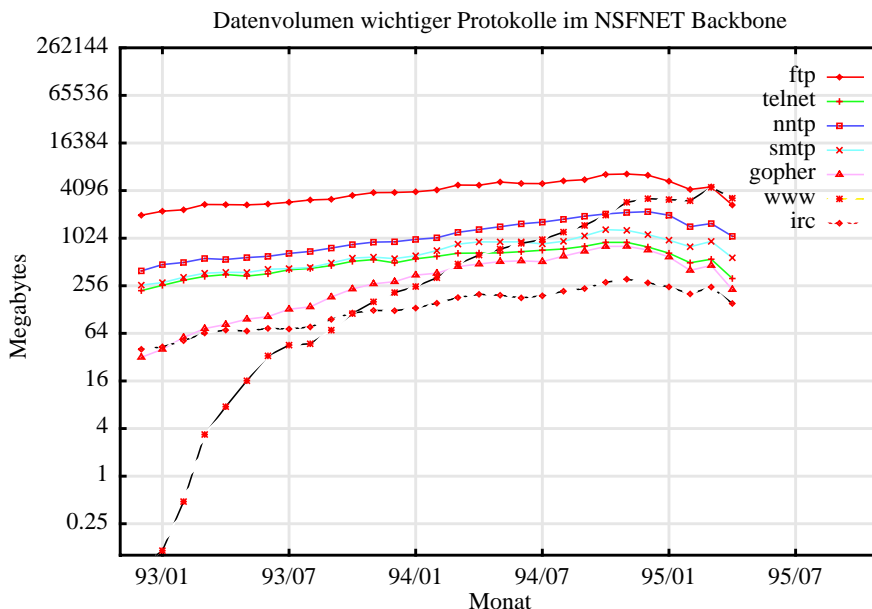


Abbildung 1: Datenverkehr im NSFNET bzgl. wichtiger Protokolle

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.

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 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 [2] wird), geht dies einher mit einer weiteren Erhöhung der Last auf alle involvierten Ressourcen¹. 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 Effekt auf.

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 und zwar als **World Wide Wait**. Jetzt standen zwar relativ einfach handhabbare Werkzeuge zur Beschaffung von Dokumenten und Da-

ten zur Verfügung, doch das Internet ist der massenhaften Abfrage und dem Transport selbiger nicht oder nur teilweise gewachsen.

Aus diesem Grunde setzen seit ca. 2 Jahren immer mehr ISP Proxycaches ein, um zumindest bzgl. WWW und FTP die Qualität ihrer Dienste zu erhöhen, Bandbreite einzusparen und damit Kosten zu minimieren.

2 Proxycaches im WWW

2.1 Zum Begriff des Proxycache

Auf der Suche nach effizienten Mechanismen zum verbesserten, schnelleren und ressourcensparenden² Zugriff auf Informationen im Internet wurde festgestellt, daß 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 geholt 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 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.

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

2. 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.

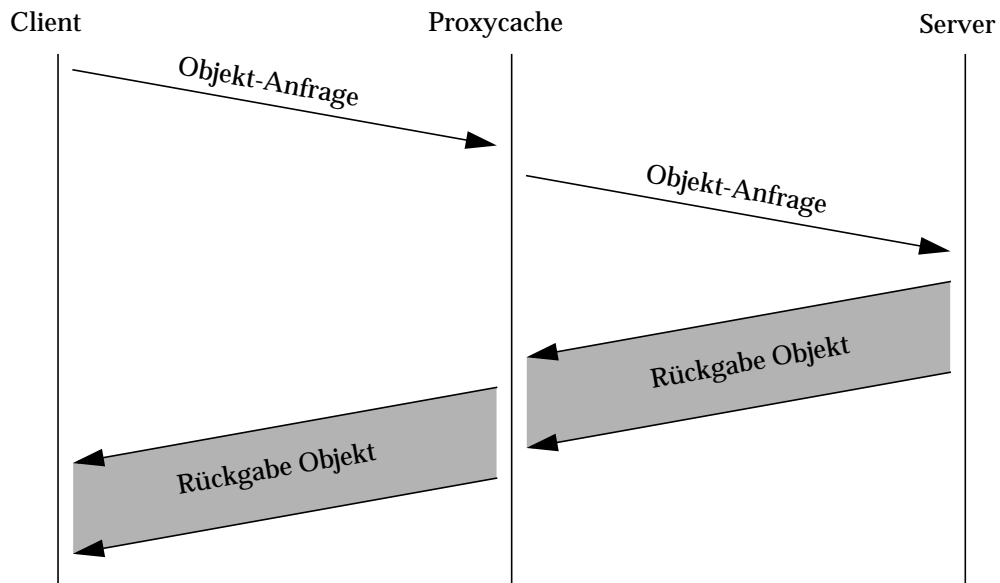


Abbildung 2: Client-Server-Kommunikation über einen Proxycache (MISS)

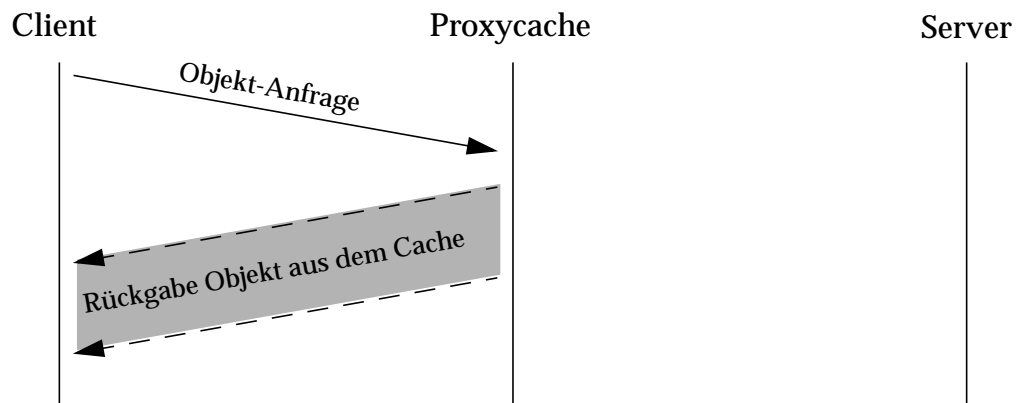


Abbildung 3: Client-Server-Kommunikation über einen Proxycache (HIT)

Befindet sich jedoch das fragliche Objekt schon im Cache (HIT, siehe Abbildung 3), 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. 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 Die Notwendigkeit eines Proxycache-Verbunds

Weitere Überlegungen sowie Forschungen im Rahmen des *Harvest research projects* [1] 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.

2.3 Die Notwendigkeit einer Proxycache-Hierarchie

Sind relativ viele Proxycaches in einem Verbund zusammengeschlossen und entspricht dessen Topologie einem vermaschten Netz, kann es ohne dem Treffen entsprechender Vorkehrungen zu einer Endlosschleife bzgl. der Anfrage nach bestimmten Objekten (Request Loop) innerhalb des Proxycache-Verbunds kommen. Moderne Softwaresysteme, wie zum Beispiel Squid [5], erkennen zwar solche Request Loops, die Suche nach der Ursache des Problems und dessen Behebung bleibt jedoch den Administratoren überlassen.

Um Request Loops vermeiden und die Anfragen nach Objekten innerhalb eines Proxycache-Verbunds besser steuern zu können, muß die Funktion aller Proxycaches in diesem Verbund "wohldefiniert" 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 derartigen Cache-Hierarchien sind im allgemeinen folgende Begriffe bzw. Funktionalitäten eines Proxycaches definiert:

- **Parent:** stellt ein Proxycache eine Anfrage an seinen Parent, muß der Parent das angeforderte Objekt zurückliefern;
- **Sibling:** stellt ein Proxycache ein Anfrage an einem Sibling, liefert dieser das angeforderte Objekt nur zurück, wenn es sich bereits in seinem Cache befindet (HIT);
- **Neighbor:** ist ein Parent oder Sibling;
- **Child:** Proxycache, der eine Anfrage an seinen Parent schickt und diesem somit unterstellt ist;
- **Root-Cache:** Parent, der in der Cache-Hierarchie auf der obersten Stufe steht, also keinen weiteren Parent besitzt.

2.4 Squid an der Universität Magdeburg

Squid [5] ist ein relativ leistungsstarker Proxycache-Server für Web-Clients, welcher HTTP-, FTP- und Gopher-Datenobjekte unterstützt. Im Gegensatz zu traditioneller Cache-Software bearbeitet Squid alle Anfragen in einem einzigen, nichtblockierenden, E/A-basierten Prozeß. Squid unterstützt das SSL-Protokoll, umfangreiche Zugriffssteuerungen und die komplette Aufzeichnung von Anfragen (Request Logging). Durch den Einsatz des leichtgewichtigen Internet Cache Protokolls (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 Systems Lookup Programm (dnsserver), einem Programm zum Herunterladen von FTP-Daten (ftpget) und einigen Client- und Management-Tools.

Seit April 1996 betreibt die Fakultät für Informatik einen Proxycache-Verbund, in dem sowohl universitätsinterne als auch -externe integriert sind. Die folgende Tabelle gibt einen schematischen Überblick über die derzeitige Struktur und die Beziehungen der einzelnen Proxycaches innerhalb des Verbundes, woraus zu ersehen ist, daß **proxycache.cs.uni-magdeburg.de** Parent

(Gateway zum Extranet) aller universitätsinterner Caches als auch anderer Magdeburger Caches ist:

... 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	3	3	3	3	3	1	1	1	1	2	2
www-cache.uni-magdeburg.de	0	0	1	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 1: Beziehungen von und zu Proxycaches in der Magdeburger Cache Hierarchie (MCH)

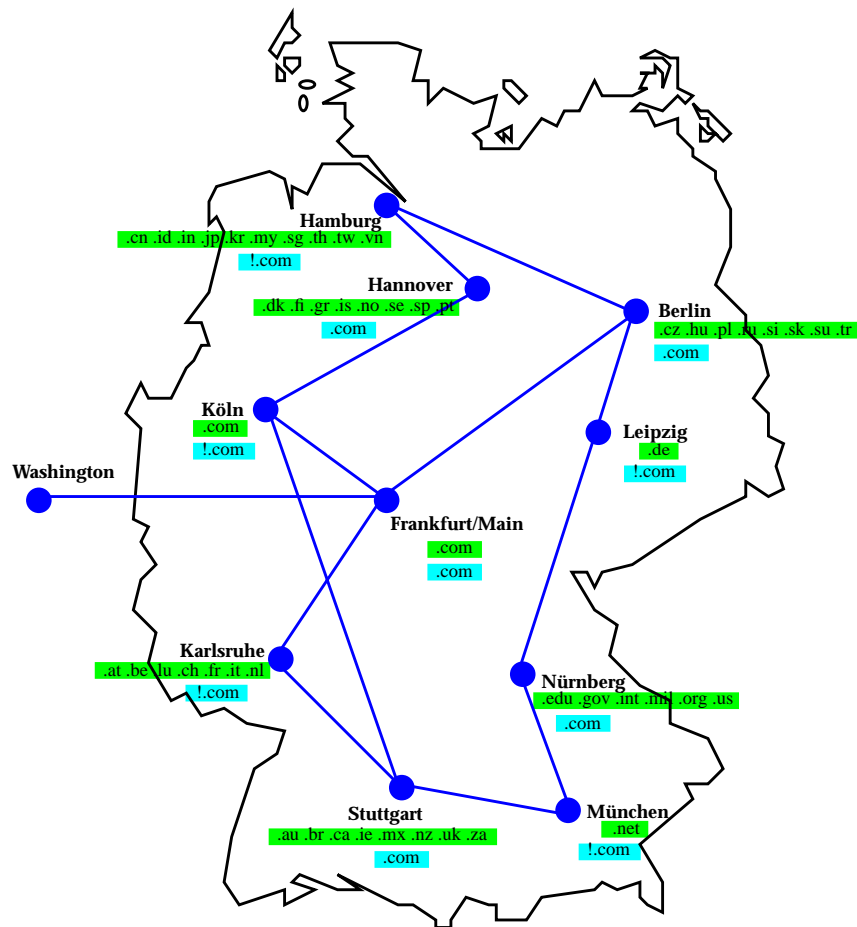
Der Proxycache.cs.uni-magdeburg.de selbst war kurzzeitig Child von verschiedenen Rootproxycaches des DFN. Leider führte dies zu einer teilweise schlechteren Quality of Service, da diese Caches partiell überlastet waren und somit das Herunterladen von Objekten bedeutend länger dauerte, als ohne deren Nutzung. Deshalb werden seit dem diese Caches nur noch als Neighbor benutzt (d. h., nur wenn sie das angeforderte Objekt im Cache haben, wird es von diesen abgefordert).

Die Ursache dafür war in der beabsichtigten Lastverteilung auf die verschiedenen Caches zu suchen. Dazu wurden alle bekannten *Topleveldomains* (TLD) auf je einen der 10 Proxycaches des DFN verteilt und ab so-

fort nur noch Objekte aus der jeweiligen Domain gespeichert (siehe Abbildung 4, grau). Dies führte zur absoluten Überlastung der für die TLD.com zuständigen Caches in Köln und Frankfurt, sowie des Caches in Leipzig (TLD.de).

Auch der DFN erkannte dieses Manko und führte ein am 16. Oktober 1997 ein neues Modell für die Lastverteilung wiederum auf Grundlage der TLD von Objekten ein:

50% aller DFN-Caches sind verantwortlich für Objekte aus der TLD.com und alle anderen für Objekte aus anderen TLDs (siehe Abbildung 4, hellgrau).



vor dem 16.10.1997
 nach dem 16.10.1997

Abbildung 4: cache_host_domain - Aufteilung in der DFN-Cache-Hierarchie

Allerdings ist diese Aufteilung nach TLDs wiederum ungünstig, da sich nun die Gesamtkapazität der Festplattenspeicher dieser Caches von ca. 60 GB (10 Caches mit je 6 GB) faktisch durch Redundanz auf ca. 12 GB reduzierte. D. h., das Lastproblem bzgl. der einzelnen Caches wurde einerseits zwar gelöst, andererseits aber auch die zur Verfügung stehende Cache-Kapazität und somit auch die Hitrate bei diesen Caches gesenkt.

Deshalb sollte nach Meinung der Autoren versucht werden, eine Lastenverteilung zu erreichen, bei der kein Objekt bzgl. aller Caches redundant gespeichert wird und somit Grundlage für eine höhere Hitrate bildet, die entsprechenden Caches aber auch nicht aufgrund dessen überlastet werden. Wie Erfahrungen zeigen, erscheint somit ein Lastenausgleich aufgrund von TLDs von Objekten nicht geeignet.

Als einen Weg für die Lösung dieses Problems könnte das Cache Array Routing Protokoll (CARP) dienen, das im folgenden kurz erläutert wird und zu dem anschließend entsprechende Untersuchungsergebnisse angegeben werden.

2.5 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 von den 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 vorgese-

nenen 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 im selbigen.

CARP basiert auf:

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

wobei die PAMT als reine ASCII-Text-Datei geführt 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 [4].

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 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.

3 Analysen zur Verbesserung der Dienstgüte im WWW

3.1 Allgemeine Verbesserungsformen

Zu den im Rahmen dieser Forschung bereits entwickelten konstruktiven Methoden (siehe <http://ivs.cs.uni-magdeburg.de/~elkner/webtools/>), wie

- die Unterstützung des Proxycache-Monitoring als Aufnahme der wichtigsten Statisti-

ken alle 5 Minuten und deren graphische Darstellung durch das Perl-Skript *gps4a.pl*,

- die Erarbeitung von Cache-Regeln für die Objektbehandlung,
- die Entwicklung und Installation eines Redirectors *jesred* beispielsweise zur Ausblendung von Werbeseiten,

werden hier vornehmlich reaktive Strategien zur Analyse auf der Grundlage gezielter Tests beschrieben, die eine permanente Systemüberwachung und -steuerung ermöglichen sollen.

3.2 Die Vorgehensweise bei den Tests und Analysen

Leider gab es bisher keine zuverlässigen Aussagen über die Qualität des Lastenausgleichs, der durch die Verwendung der oben genannten Verfahren 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 Untersuchungen die Hauptroutine von Squid umprogrammiert und durch geeignete, neu programmierte Module ergänzt und somit zum Testprogramm namens *lb-checker* für verschiedene Lastenausgleichs-Algorithmen umfunktioniert (siehe <http://ivs.cs.uni-magdeburg.de/~elkner/webtools/>). Da eine Evaluierung im Online-Betrieb zu aufwendig wäre und keine Langzeit-Aussagen zuließe, wurden als Datenquellen die „access_log“-Dateien¹ [6] 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 in gleicher Weise wirken. Daher 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)

-
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

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 2: Parameter für Load balancing Test: CARP

und analysiert. Dabei wurde 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 Untersuchungen dienen.

3.3 Die Tests zu CARP und Squid-CARP

In Test 1 wurde das CARP-Protokoll, in Test 2 das leicht modifizierte CARP-Protokoll von Squid Version $\geq 1.1b25$ (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 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 zeigen (siehe die Abbildungen A1 bis A3 im Anhang), 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 führte zu der Vermutung, 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 den Abbildungen A4 bis A8 im Anhang 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 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.

3.4 Die Anwendung des Simple Hashes

Ausgehend von CARP und dem im Super Proxy Script [6] dargestellten Algorithmus wurde von den 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 wurde anschließend auf den Bereich 0..100 abgebildet, dem er mit 100 multipliziert und anschließend die Summe aller bisher so ermittelten Produkte aufaddiert wurde (siehe Tabelle 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 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 jeweiligen 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 zurückgegeben.

```

{
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 %ld\n",
                                tp->host,
                                url_hash);
        return tp;
    }
}
}

```

```

peer *
fileSelectPeer(request_t * request)

```

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/
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: 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 langfristig eine Verteilung der Last auf die Proxycaches im Bereich $\pm 5\%$ 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.

4 Schlußfolgerungen und Ausblick

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 Forschungsaufgabe bleiben, hier nach besseren, effizienteren Lastausgleichsverfahren zu suchen, um weiter die Qualität des WWW-Dienstes zu verbessern sowie auch die Kosten dafür relativ niedrig zu halten bzw. zu senken.

Ein Ansatzpunkt für weitere Untersuchungen wäre zum Beispiel die Analyse bisheriger Access-log-files bzgl. des Domain-Namens 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, Anzahl der Anfragen allgemein und Mustererkennung (pattern matching) in URLs.

Um generell einen Überblick bzgl. der genannten statistischen Werte zu haben, wurde bereits ein Programm namens *jesalfa*¹ entwickelt, welches Access-log-files 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* gewonnenen Ergebnisse, 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.

1. siehe <http://ivs.cs.uni-magdeburg.de/~elk-ner/webtools/jesalfa/>

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 sowie 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 Literatur

- [1] Bowman et al.: *The Harvest Information Discovery and Access System*. Internet Research Task Force, <http://harvest.transarc.com>
- [2] Kindel et al.: *Inserting objects into HTML*. W3C Working Draft, Februar 1997
- [3] Merit Network Inc.: *GVU's NSFNET Backbone Statistics*. December 1992 - April 1995
- [4] Valloppillil, V.; Ross, K.W.: *Cache Array Routing Protocol v 1.0*. University of Pennsylvania, Februar 1998
- [5] Wessels, D.: *The Squid Internet Object Cache*. National Laboratory for Applied Network Reserach, <http://squid.nlanr.net>
- [6] White Paper: *Super Proxy Script - How to make distributed proxy servers by URL hashing*. Sharp Corporation, August 1996 - 1998

Anhang: Ausgewählte Tests zum Lastenausgleich

Die folgende Angabe ausgewählter Messungen soll lediglich die prinzipielle Vorgehensweise verdeutlichen. Grundlage bilden dabei 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 Neig-

hbors und der Maschine, auf dem das PCMS läuft, bei den Analysen nicht berücksichtigt. Zeitraum jeder einzelnen Messung ist Monatsbeginn bis Monatsende. Die linke Seite zeigt den jeweiligen CARP-Test, während das rechte Bild den Squid-CARP-Test beinhaltet.

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 T1: Load Balancing (LB) CARP, Squid-CARP: Parameter

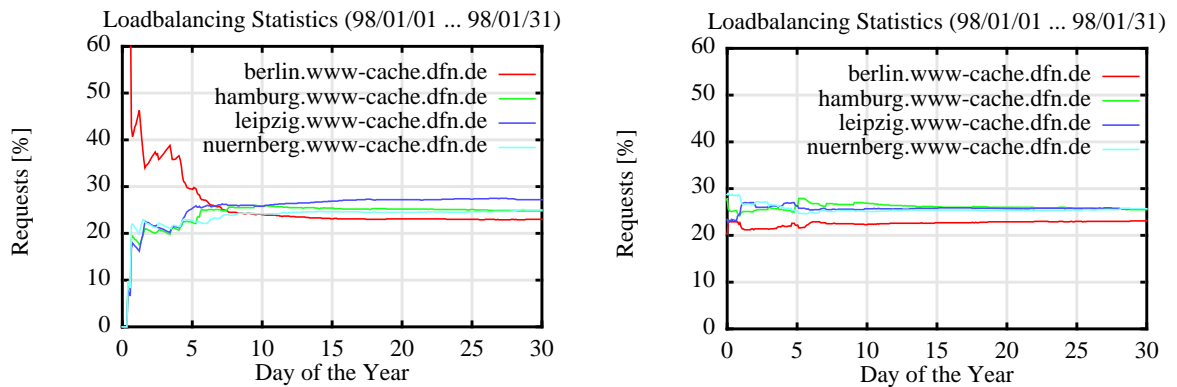


Abbildung A1: LB (absolut): CARP (left), Squid-CARP (right); total: 724.091 requests

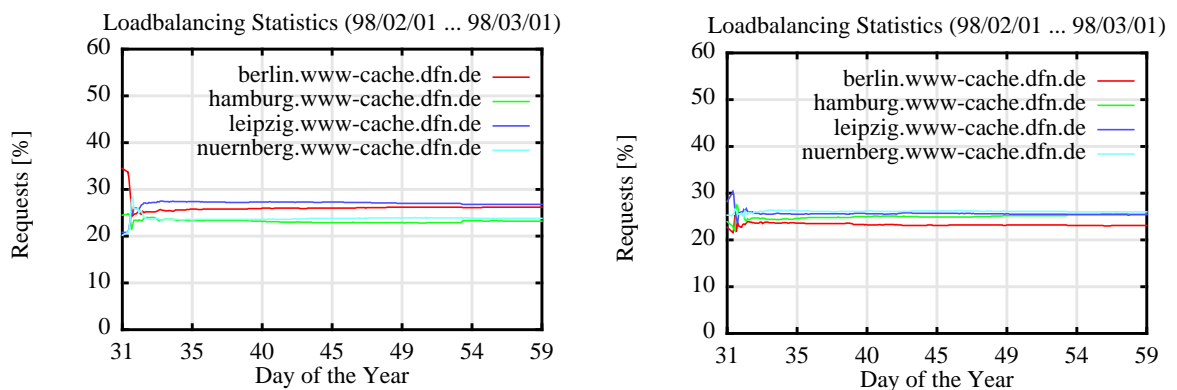


Abbildung A2: LB (absolut): CARP (left), Squid-CARP (right); total: 813.652 requests

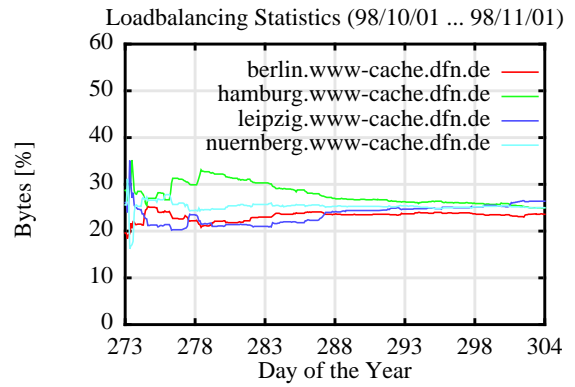
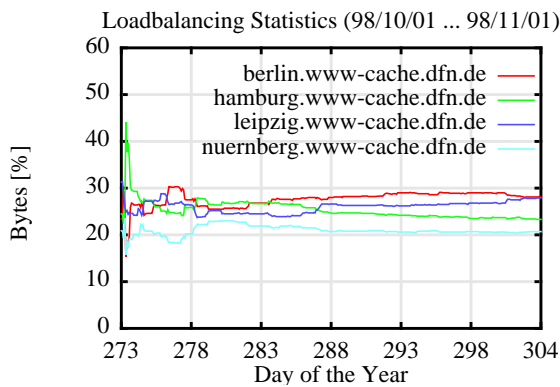
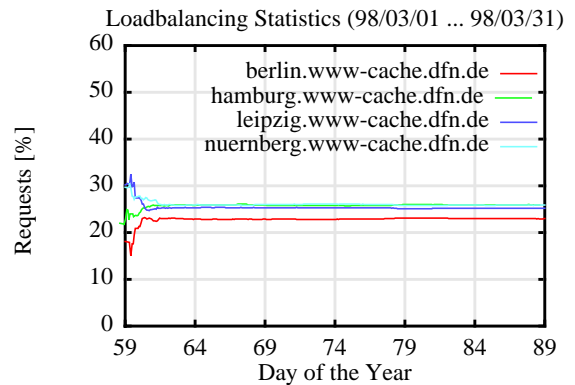
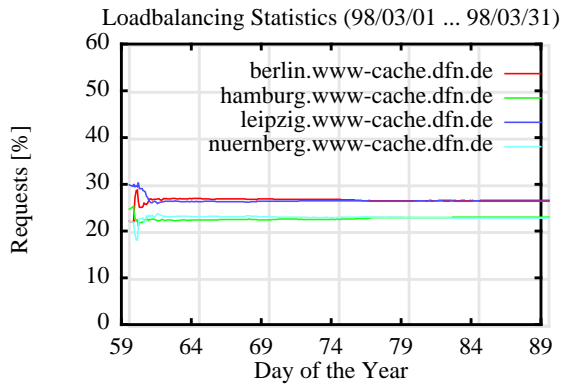


Abbildung A3: LB (absolut): CARP (left), Squid-CARP (right); total: 13.249.527.918 bytes

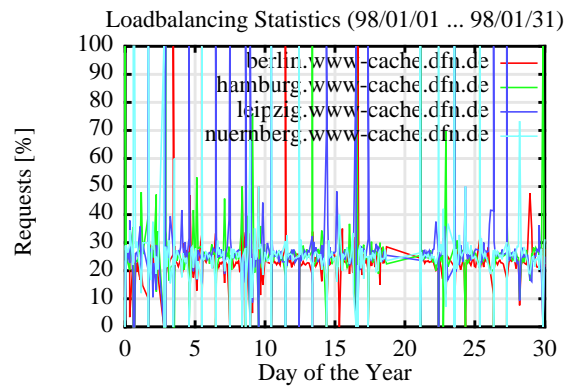
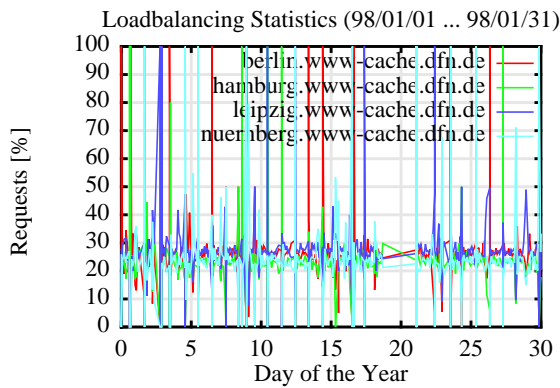


Abbildung A4: LB (relativ): CARP (left), Squid-CARP (right); total: 724.091 requests

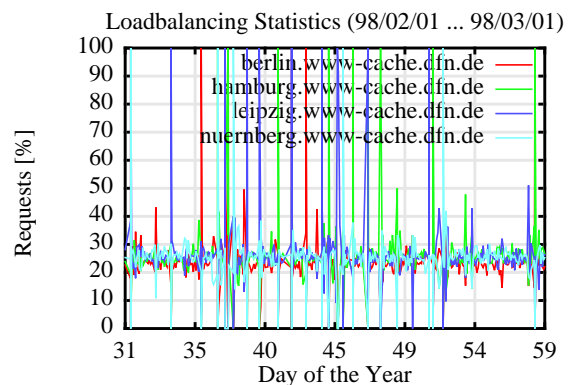
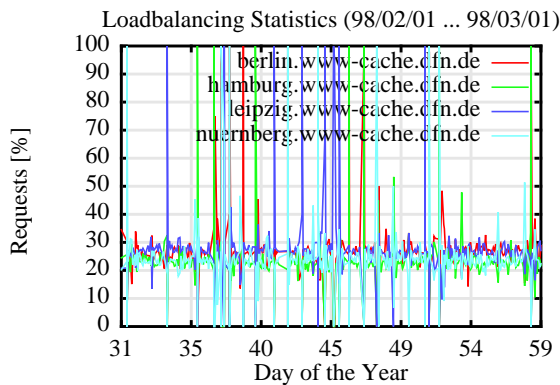


Abbildung A5: LB (relativ): CARP (left), Squid-CARP (right); total: 813.652 requests

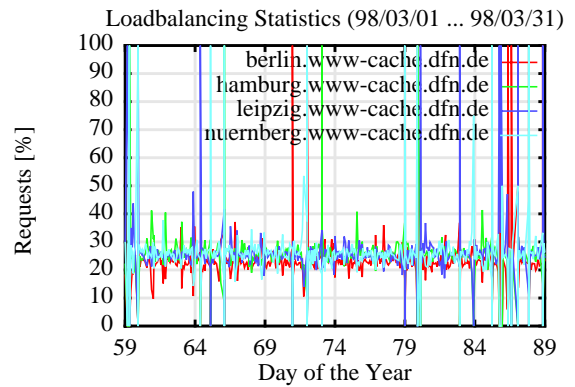
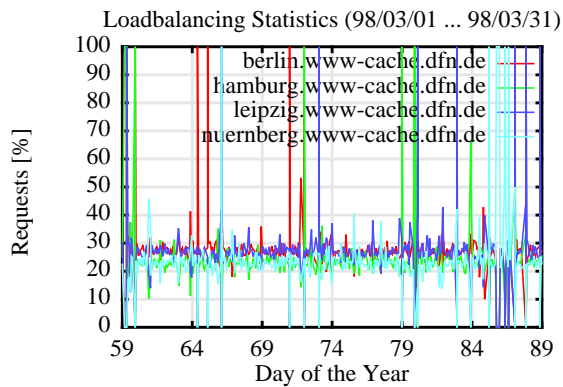


Abbildung A6: LB (relativ): CARP (left), Squid-CARP (right); total: 797.286 requests

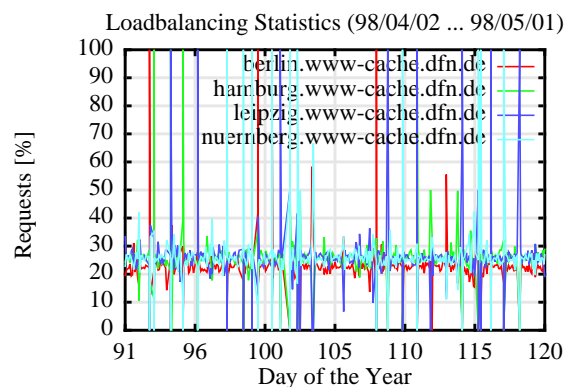
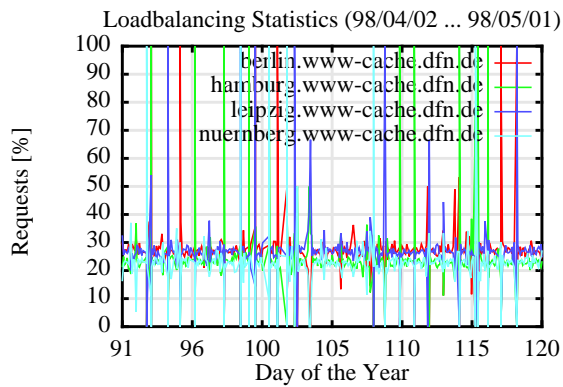


Abbildung A7: LB (relativ): CARP (left), Squid-CARP (right); total: 1.012.006 requests

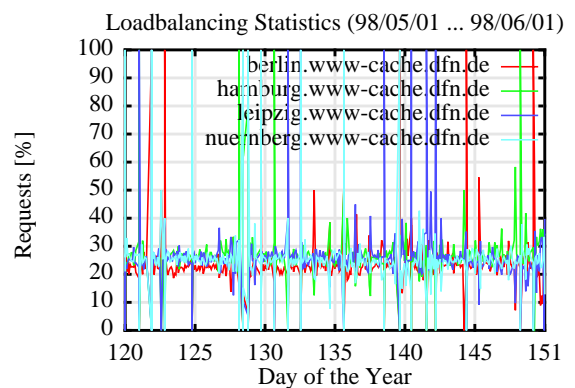
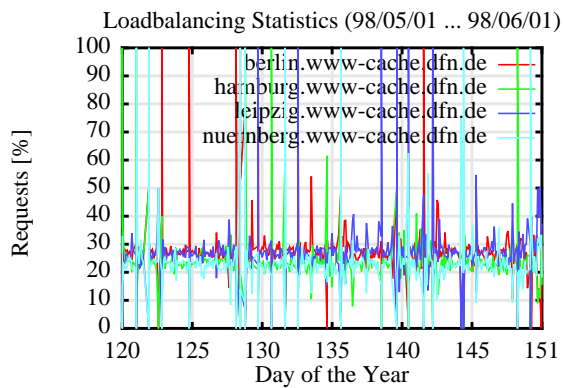


Abbildung A8: LB (relativ): CARP (left), Squid-CARP (right); total: 1.065.412 requests

