

2 Prinzip eines Proxys

Das folgende Kapitel beschreibt die grundsätzliche Funktionsweise eines Proxyserver, seine Vor- und Nachteile sowie verschiedene technische Ansätze einer Kommunikation zwischen Proxyservern.

2.1 Was ist ein Proxy?

Proxy heißt auf Deutsch so viel wie *Bevollmächtigter* oder *Stellvertreter*. Ein Proxyserver hat also in erster Linie die Aufgabe, *stellvertretend* für einen Client eine Anfrage an einen Server zu stellen, die Antwort vom Server entgegenzunehmen und sie dann an den anfragenden Client weiterzuleiten.

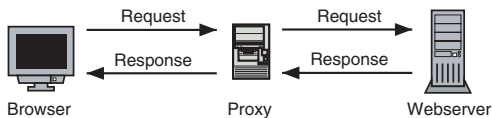


Abbildung 2.1
Client-Proxy-Server-Prinzip

Dabei wird die Verbindung vom Client (Browser) zum Webservice vollständig getrennt. Der Client fragt den Proxy (Request) und bekommt seine Antwort vom Proxy (Response). Der Proxy fragt den Webservice (Request) und bekommt die Antwort vom Webservice (Response).

Es besteht somit keine direkte Verbindung mehr zwischen Client und Webservice. Beide können in völlig unterschiedlichen Netzen stehen, und der Webservice wird (je nach Konfiguration des Proxys) nichts von der Identität des Clients erfahren.

2.2 Warum brauche ich einen Proxy?

Auf den ersten Blick ist ein Proxyserver also ein zusätzliches – scheinbar unnötiges – Glied in der Kette zwischen Client (Browser) und (Web-)Server.

Diese Stellvertreterfunktion eines Proxys ist jedoch aus mehreren Gründen sinnvoll:

- ❑ **Sicherheit**
Der Client fragt nicht selbst, er lässt seinen Proxy fragen. Der Client tritt damit z. B. bei Anfragen ins Internet nicht selbst in Erscheinung. Die Antwort eines Servers geht ebenfalls nur an den Proxy, nicht an den Client. Ein damit verbundener Angriff würde also in erster Linie den Proxy und nicht den Client treffen. Ein Proxy kann also den Client schützen.
- ❑ **Verkehrslenkung**
Wenn Daten aus unterschiedlichen Netzen geholt werden müssen, kann ein Proxy zu einer vereinfachten Verkehrslenkung beitragen. Es muss nicht jeder Client wissen, was er wo herbekommt. Die Regeln hierfür können allein im Proxy gepflegt werden.
- ❑ **Zugriffssteuerung**
Ein Proxy ist auch geeignet, um Zugriffe zu steuern und zu regeln. Nicht jeder darf überallhin? Für bestimmte Zugriffe soll eine Authentifizierung vorgenommen werden? Dies kann ein Proxyserver erledigen.
- ❑ **Protokollierung**
Jeder Zugriff über einen Proxyserver kann protokolliert werden. Proxy-Protokolle können zur Statistik, Abrechnung oder Kontrolle der Zugriffe ausgewertet werden. Hierzu gibt es ähnlich wie bei Webservern unterschiedliche Auswertungsprogramme.

2.3 Cache-Proxy

*Cache = Puffer,
(Zwischen-)Speicher,
Lager*

Neben dieser Stellvertreteraufgabe haben viele Proxyserver wie auch Webclients eine *Cache*-Funktionalität. Das heißt, ein Proxyserver holt das angeforderte Objekt vom Webserver, liefert es an den Client aus und behält eine Kopie des Objekts in seinem eigenen Speicher.

Diesen Zwischenspeicher nennt man *Cache*. Den Proxy, der dies tut, nennt man einen Cache-Proxy, Proxy-Cache oder einfach nur *Cache*.

Fordert jetzt ein zweiter Client bei einem Cache-Proxy das gleiche Objekt an, kann der Proxyserver dieses Objekt aus dem eigenen Speicher liefern und muss nicht erst den (entfernten) Webserver befragen. Der Client selbst kann ebenfalls einen Cache anlegen und Objekte zwischenspeichern, um sie nicht erneut vom Proxy- oder Webserver holen zu müssen.

2.3.1 Warum einen Cache-Proxy einsetzen?

Ein Cache-Proxy ist ohne Zweifel teurer als ein reiner Proxyserver. Es wird eine erheblich größere Plattenkapazität, mehr Speicher und ggf. auch ein leistungsfähigerer Prozessor benötigt.

Folgende Argumente können diese Investitionen trotzdem rechtfertigen:

- ❑ Beschleunigung
Da der Weg zwischen Proxy und Client meist kürzer ist als der Weg zwischen Webserver und Client, sind auch die Antwortzeiten des Proxys aus dem eigenen Cache meist deutlich kürzer. Abgesehen von Webservern im lokalen Netz kann je nach Entfernung und Anbindung zum Webserver die Antwortzeit aus dem Cache realistisch um den Faktor 2 bis 100 schneller sein als direkte Zugriffe.
- ❑ Bandbreite
Durch die Datenhaltung im Cache werden viele Anfragen an den Zielservers überflüssig. Der Proxy liefert die Antwort aus dem Cache, anstatt sie über eine (externe) Verbindung zu holen. Der Verkehr zwischen Proxy und Client ist zwar unverändert, aber die Verbindung vom Proxy zum Webserver kann deutlich entlastet werden. Realistische Einsparungen in der Bandbreite liegen – abhängig vom Nutzungsverhalten – bei etwa 20 bis 50 Prozent.
- ❑ Verfügbarkeit
Bei unsicheren Verbindungen oder schlechter Verfügbarkeit externer Webserver kann ein Proxyserver u. U. auch zu einer Erhöhung der Verfügbarkeit dieser Inhalte führen. Wurde ein Objekt einmal im Cache abgelegt, kann es bei Ausfall der Verbindung oder des externen Webserver ggf. noch aus dem Cache geliefert werden.

Die meist eher geringen Mehrkosten für die Hardware und Konfiguration eines Cache-Proxys werden also u. U. durch eine Reduzierung der nötigen Bandbreite, eine Beschleunigung der Zugriffe und ggf. eine höhere Verfügbarkeit mehr als aufgehoben.

2.3.2 Warum keinen Cache-Proxy einsetzen?

Neben den vielen eindeutigen Vorteilen von Cache-Proxys gibt es jedoch auch gute Gründe, die gegen einen Cache-Proxy sprechen. In einigen Fällen kann es durchaus sinnvoll sein, auf eine Cache-Funktionalität zu verzichten:

- ❑ Kosten eines Cache
Wie oben schon genannt, benötigen Sie für einen Cache entsprechende Speicherkapazität, sowohl als Festplatte wie auch im Hauptspeicher. Der Hauptspeicherbedarf Ihres Proxys wächst proportional zum Cache-Plattenplatz.
- ❑ Verzögerungen durch einen Cache
Ein Cache spart zwar in den meisten Fällen Zeit, die Suche eines Objekts in einem Cache(-Verbund) erfordert aber selbst wiederum Zeit, die mit der Größe des Cache zunimmt. Nutzen Sie den Proxy nur für Webserver in einem lokalen Netz mit guter Bandbreite, kann dieser Effekt unter sehr ungünstigen Umständen sogar zu geringfügig längeren Antwortzeiten führen als bei einem direkten Zugriff.
- ❑ Aktualität
Die Frage »Wie aktuell ist das Objekt, das ich im Cache habe?« kann ein starkes Argument gegen einen Cache sein. Nicht jedes Objekt eignet sich für einen Cache. Moderne Proxyserver schließen zwar solche Objekte weitgehend aus dem Cache aus oder prüfen regelmäßig die Aktualität, jedoch kann es trotzdem zur Auslieferung veralteter Objekte kommen. Ist eine laufende Aktualität aller Informationen für Sie ein zwingendes Argument, ist evtl. von einem Cache abzuraten.
- ❑ Rechtliche Probleme
Die vom Client angeforderten Objekte werden im Cache gespeichert. Hier könnten je nach aktueller Rechtslage Probleme in Bereichen wie Urheberrecht (für geschützte Veröffentlichungen, Musik etc.) oder Strafrecht (illegale Inhalte wie z. B. Pornografie oder Rechtsradikalismus) auftreten. Zurzeit ist die Rechtslage in Deutschland zwar eher *Cache-freundlich*, da hier ja nicht bewusst gehandelt wird, aber Rechtslagen können sich bekanntlich auch ändern.

Nicht alle Argumente müssen in jedem Fall greifen. Sie müssen die Vor- und Nachteile eines Caches genau abwägen. In vielen Fällen wird es sinnvoll sein, einen Cache einzusetzen. Besonders wenn viele statische Daten über langsame Verbindungen, z. B. aus dem Internet, geholt werden müssen, kann ein Cache deutliche Vorteile bringen.

Nutzen Sie jedoch überwiegend Daten aus dem eigenen (schnellen) Netz, die noch dazu überwiegend dynamisch erzeugt werden und sich damit laufend ändern, kann sich ein Cache im ungünstigsten Fall sogar durch geringfügig längere Antwortzeiten oder veraltete Daten negativ bemerkbar machen.

2.4 Das Cache-Verhalten beeinflussen

Das Cache-Verhalten eines Proxys kann – neben der eigenen Konfiguration – sowohl vom Client als auch vom Webserver beeinflusst werden. Für einen gut funktionierenden Cache müssen die Betreiber der Webserver und die Webdesigner auch *Cache-freundliche* Seiten erstellen, und die Clients müssen entsprechend konfiguriert sein, um einen Cache-Proxy auch optimal zu nutzen.

*Cache-freundliche
Webseiten*

2.4.1 Webserver

Webserver können beim Ausliefern eines Objekts zusätzliche Informationen in einem Header senden [1.1.5].

Für den Proxyserver sind u. a. folgende Header-Daten von besonderem Interesse:

*Header = Kopfzeile,
Kennsatz, Titel,
Leitvermerk*

Last-Modified: liefert das Datum der letzten Änderung des Objekts.

Dieser Wert wird u. a. für die Berechnung benötigt, wann ein Objekt als nicht mehr aktuell angesehen wird.

Expire: bestimmt, wann das Objekt abläuft. Nach *Expire* muss das Objekt vom Proxy auf seine Aktualität geprüft werden.

Cache-Control: weist den Proxy an, wie er sich bei diesem Objekt zu verhalten hat, z. B. ob er es überhaupt in den Cache aufnehmen darf.

Beispiel:

```
Cache-Control: max-age=86400, must-revalidate
Expires: Tue, 30 Jul 2002 19:43:45 GMT
Last-Modified: Thu, 30 May 2002 01:31:01 GMT
```

2.4.2 Webautoren

Webautoren oder -designer haben bei der Seitenerstellung und beim Webdesign auch erheblichen Einfluss auf die Cache-Freundlichkeit ihrer Seiten.

Insbesondere bei Skripten werden meist keine Header-Daten generiert, so dass ein Proxy oder Browser keine weiteren Informationen über die Aktualität der erhaltenen Daten bekommt und diese nicht im Cache ablegt.

Klickt der Anwender dann beim Surfen mit dem Button [Back] oder [Zurück] erneut über eine Skriptseite, werden auch die gleichen Daten erneut angefordert.

Es wird sicher einige Skripte geben, die wirklich *dynamisch* sind, d. h., dass sie bei jedem Aufruf auch ein anderes Ergebnis liefern. Erfahrungsgemäß werden jedoch in den meisten Skripten die gleichen Daten erscheinen. Es wäre also – je nach Daten – eine *Haltbarkeitszeit* von fünf Minuten bis zu einer Stunde durchaus vertretbar. Das würde bewirken, dass ein *Klicken* über dasselbe Skript dieselben Daten aus dem Cache liefern könnte und nicht der Webserver bemüht werden müsste, die Daten erneut zu generieren und zu senden.

Beispiel: Ein Cache-Control: max-age=600 im Header der gesendeten Daten würde diese zehn Minuten lang im Proxy oder Browser speichern.

2.4.3 Client

Der Client ist ebenfalls in der Lage, durch die Formulierung seiner Anfrage das Cache-Verhalten des Proxys zu beeinflussen.

Drückt der Anwender beispielsweise auf [STRG] + [Reload]/[Aktualisieren] in seinem Browser, sendet dieser in seiner Anfrage an den Proxy ein Pragma no-cache mit, was den Proxy veranlasst, nicht den Cache zu durchsuchen, sondern das Objekt direkt vom Webserver zu holen.

Viele Browser verfügen darüber hinaus über einen eigenen Cache, den sie je nach Konfiguration vor einer Anfrage an den Proxyserver nutzen.

2.4.4 Interessenkonflikt

Die Möglichkeiten, das Cache-Verhalten zu manipulieren, sind vielfältig. Für jede dieser Möglichkeiten gibt es durchaus sinnvolle Anwendungen. Die Fülle der Möglichkeiten führt jedoch auch manchmal zu unerwünschten Nebeneffekten.

Technische Vorteile

Der Betreiber eines Cache-Proxys verfolgt i. d. R. das Ziel, seine Zugriffe zu beschleunigen und Bandbreite im Netz einzusparen. Ein einmal geholtes Objekt muss bei Folgeanfragen nicht mehr vom entfernten Webserver geholt werden, sondern wird aus dem schnelleren Cache geliefert.

Auch seriösen Webserver-Betreibern kommt dies normalerweise entgegen, da auch hier Bandbreite gespart wird, wenn nicht jede Anfrage vom Webserver selbst beantwortet werden muss.

Marketing-Nachteile

Nun gibt es im Internet jedoch auch viele werbefinanzierte Angebote. Ein Webserver-Betreiber, der sich über Werbung finanziert, muss seinen Werbeplatz auch irgendwie vermarkten. Ein Webserver, der häu-

fig angefragt wird, ist für Werbepartner interessanter als einer, der vielleicht gerade mal zehn *Klicks* am Tag hat.

Also sind solche Webserver-Betreiber – obwohl es durchaus auch andere Lösungsansätze gibt – nicht selten daran interessiert, möglichst viele Anfragen auf ihrem Webserver zu protokollieren. Sie werden also jede sich bietende Möglichkeit nutzen, einen Proxyserver zu umgehen.

Auch der verstärkte Einsatz von dynamischen Seiten und Datenbanken schränkt die Cache-Möglichkeiten eines Proxys immer weiter ein. Meiner Erfahrung nach sind – je nach Nutzungsverhalten – nur noch etwa 30–50% aller Objekte im Internet cachebar. Die Tendenz ist hierbei klar abnehmend.

2.5 Cache-Proxy-Verbund

Für den Betrieb größerer Netze ist *ein* Cache-Proxy allein irgendwann nicht mehr ausreichend. Es kommen weitere dazu, entweder parallel zur Lastverteilung oder über das Netz verteilt, um Bandbreite zu reduzieren.

Dies stellt grundsätzlich kein Problem dar, Proxys können prinzipiell beliebig parallel betrieben oder hintereinander geschaltet werden. Sie benötigen i. d. R. keine gemeinsame Datenbasis. Um jedoch effizient zu arbeiten, wäre eine solche wahllose Zusammenschaltung nicht sehr sinnvoll. Es wäre besser, den Cache der Proxys in einem Verbund zusammenzuschalten und ihn gemeinsam zu nutzen.

Allerdings ergeben sich daraus schon ein paar grundlegende Fragestellungen. Wenn ein Proxy eine Anfrage von seinem Client bekommt und er das angefragte Objekt nicht selbst im Cache hat, muss er prüfen können, ob:

- ein anderer Proxy dieses Objekt vorhält,
- falls mehrere Proxys dasselbe Objekt vorhalten, welcher es am schnellsten liefern kann
- falls kein anderer Proxy das Objekt vorhält, welcher Proxy das Objekt am schnellsten herbeischaffen kann,
- oder ob es sinnvoller ist, das Objekt selbst zu besorgen.

Hierzu werden die Cache-Proxys erst einmal eingruppiert.

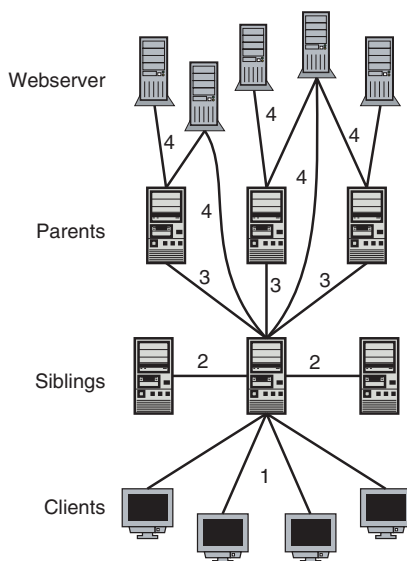
Alle Cache-Proxys, die ein Proxy kennt, sind seine Nachbarn (*Neighbours*). Die Nachbarn unterscheiden sich dabei in *Siblings* (Geschwister) und *Parents* (Eltern).

Siblings sind Nachbarn, die mit dem anfragenden Proxy selbst auf gleicher Ebene stehen. Das heißt, sie bedienen i.d.R. selbst Clients und haben ihrerseits übergeordnete Proxys (Parents). Siblings können normalerweise nur Objekte liefern, die sie selbst im Cache haben. Sie werden jedoch keine Anfragen entgegennehmen, für die sie selbst direkt beim Webserver fragen müssten.

Parents sind übergeordnete Proxys (Eltern), die i.d.R. weitergehende Verbindungen haben als der anfragende Proxy selbst. Parents können Objekte liefern, die sie selbst im Cache haben, sie können aber auch Anfragen für unbekannte Objekte entgegennehmen, die sie selbst holen oder weiterleiten.

Abbildung 2.2 verdeutlicht die Verbindungstypen und die Klassifizierung der Nachbarn.

Abbildung 2.2
Verbindungstypen in
einem Cache-Verbund



Die erste Beziehung eines Proxys ist die zu seinen Clients (1). Von ihnen bekommt er die Anfragen. Zur Beantwortung dieser Anfragen hat er die Möglichkeit, seine Nachbarn zu befragen. Die Siblings stehen ihm dabei nur mit ihren eigenen Caches zur Verfügung (2). Von ihnen kann er Objekte bekommen, die diese bereits auf anderem Wege erhalten haben. Ebenso kann er die Parents befragen (3). Diese können ihm sowohl aus ihrem eigenen Cache antworten, als auch weitergehende Anfragen für ihn stellen (4). Gegebenenfalls kann der Proxy auch einige Ziele direkt erreichen (4), ohne einen Parent zu befragen.

Die Vorgehensweise des Proxys ist dabei folgende:

- ❑ Der Proxy bekommt eine Anfrage von einem Client.
- ❑ Er prüft, ob er das angefragte Objekt im eigenen Cache hat. Wenn ja (und noch aktuell), wird er es ausliefern.
- ❑ Wenn nicht, muss er prüfen, ob das Objekt bei seinen Nachbarn (Siblings oder Parents) im Cache vorhanden ist. Wenn ja, kann er das Objekt vom jeweiligen Nachbarn beziehen.
- ❑ Wenn nicht, muss er prüfen, ob er das Objekt selbst besorgen kann. Wenn ja, wird er dies tun.
- ❑ Wenn nicht, muss er einen der Parents befragen, der in der Lage ist, ihm das angefragte Objekt zu beschaffen.

Dies ist eines von vielen Verbundmodellen. Andere Modelle arbeiten mehr nach dem Prinzip von Lastverteilung. Dabei werden z. B. bestimmten Proxys bestimmte Domain-Räume zugewiesen, für die sie zuständig sind, oder Anfragen werden nach einer Art *Round-Robin* verteilt.

2.6 Internet Cache Protocol (ICP)

ICP (Internet Cache Protocol) stammt noch aus den Ursprüngen von Squid und wurde 1997 in die RFCs [8, 9] aufgenommen.

Es basiert auf dem UDP-Protokoll und dient in erster Linie dazu, die Verfügbarkeit eines Objekts auf Nachbar-Proxys zu ermitteln. ICP bestimmt nebenbei anhand der Antwortzeit der Nachbarn auch, wie gut oder schlecht diese zu erreichen sind.

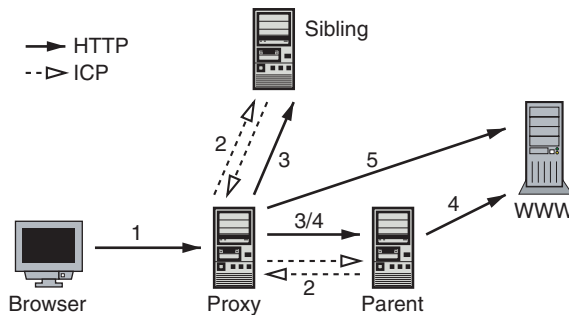


Abbildung 2.3
Verbindungen des
Internet Cache Protocol

Dazu wird nach einer Anfrage von einem Client (Abbildung 2.3, 1) erst einmal an alle Nachbarn (egal ob Sibling oder Parent) ein ICP-QUERY mit dem URL des gesuchten Objekts gesendet (2).

ICP-Query

Jeder Nachbar wird dann eine entsprechende Antwort senden. Dies kann eine Meldung sein, die besagt, dass das Objekt vorhanden ist (UDP_HIT), nicht vorhanden ist (UDP_MISS) oder dass ein Fehler aufgetreten ist (2). Die Liste aller UDP-Antworten finden Sie in Anhang A.

Der Proxy reagiert dann je nach Art und Anzahl der eingegangenen Antworten.

Wird ein Treffer (UDP_HIT) empfangen, so wird die Anfrage unverzüglich per HTTP an den betreffenden Nachbarn weitergeleitet (3).

Wurde nach dem Eingang aller Antworten oder nach der als `icp_query_timeout` vorgegebenen Zeit (i.d.R. maximal zwei Sekunden) kein Treffer erzielt, wird die Anfrage normalerweise an den Parent-Nachbarn weitergeleitet, der als Erster ein UDP_MISS zurückgesendet hat (FIRST_PARENT_MISS). Das heißt, es wird der Parent-Proxy befragt, der am schnellsten geantwortet hat (4).

Ist kein Parent definiert oder hat keiner der angegebenen Parents geantwortet, wird Squid versuchen, die Anfrage direkt an den Webserver zu stellen (5).

ICP hat jedoch zwei entscheidende Nachteile:

- ❑ Wenn ein Nachbar nicht antwortet, werden die Anfragen aufgrund des `ICP_QUERY_TIMEOUT` verzögert. Dies kann bei der Standardvorgabe bis zu zwei Sekunden sein. Zwar gibt es auch hier Mechanismen, die verhindern, dass z. B. ein ausgefallener Nachbar alle Anfragen verzögert, aber gerade in einem großen Cache-Proxy-Verbund, in einem instabilen Netz oder bei langen Antwortzeiten einiger Nachbarverbindungen kann ICP zu einer eher negativen Gesamtperformance führen.
- ❑ Bei hochfrequentierten Proxy-Verbänden, mit vielen Nachbarbeziehungen, kann der ICP-Verkehr einen erheblichen Anteil am Gesamtverkehr ausmachen. Zwar sind die einzelnen Anfragen und Antworten sehr klein, können aber u. U. ein Vielfaches der HTTP-Anfragen ausmachen.

Der Einsatz von ICP kann unter bestimmten Rahmenbedingungen sinnvoll sein, besonders da es sehr leicht zu implementieren ist. Voraussetzung sollte jedoch eine nicht zu hohe Anzahl von Nachbarbeziehungen und ein sauber funktionierendes schnelles Netz sein.

2.7 Cache Digests

Cache Digests sind Cache-Objekte, die ein Inhaltsverzeichnis des gesamten Caches enthalten.

Um den Cache Digest möglichst kompakt zu halten, wird er sehr hoch und *nicht* verlustfrei komprimiert. Es sind nicht die vollständigen URLs der im Cache enthaltenen Objekte abgelegt, sondern ein nochmals komprimierter MD5-Hash-Wert über jeden URL.

Dieser Cache Digest wird – sofern diese Funktion beim Kompilieren eingeschaltet wurde – beim Start von Squid im RAM erzeugt und wie ein gewöhnliches Objekt im Cache abgelegt. Ein Nachbar-Cache kann dieses Objekt per ganz normalem HTTP-Request anfordern.

Wird nun an einen Proxy im Verbund eine Anfrage gestellt, wird zuerst über den angefragten URL ein MD5-Hash-Wert gebildet. Dieser Hash wird mit dem Inhalt der Cache Digests aller bekannten Nachbarn verglichen. Wird eine Übereinstimmung in irgendeinem Cache Digest gefunden, so wird der URL an den betroffenen Proxy per HTTP angefragt.

Da die Digests nicht verlustfrei komprimiert wurden, gibt es allerdings auch noch die Möglichkeit, dass zwar der Hash mit einem Eintrag in einem Digest übereinstimmt, der URL zu diesem Hash aber ein anderer ist, der nur zufällig das gleiche Ergebnis wie für den angefragten URL ergibt. In diesem Fall kann die Anfrage vom Nachbar-Proxy nicht beantwortet werden und er gibt stattdessen ein `CACHE_DIGEST_MISS` zurück. Der anfragende Proxy muss das Objekt dann selbst holen.

Die Größe eines Digest in Byte wird nach der folgenden Formel berechnet:

$$digest_size = int(\frac{capacity \cdot bits_per_entry + 7}{8})$$

2.8 Cache Array Routing Protocol (CARP)

Das Cache Array Routing Protocol (CARP) wurde 1998 entwickelt und im Microsoft Proxyserver eingesetzt. Im Gegensatz zu ICP und Cache Digest findet hier kein Datenaustausch zwischen den Proxyservern statt.

Es dient eher der besseren Skalierbarkeit von Proxyservern. Dabei wird der gesamte URL-Bereich über eine Hash-Funktion auf mehrere Proxyserver verteilt. Der Zusammenschluss dieser Proxyserver wird dabei als Cache Array bezeichnet.

Durch die Aufteilung der URLs wird verhindert, dass Objekte mehrfach im Cache verschiedener Proxys gehalten werden. Gleichzeitig wird auf diese Weise eine Art Lastverteilung über alle Proxys im Cache Array erreicht.

CARP ist also eher für einen Cluster-ähnlichen Zusammenschluss von Proxyservern konzipiert.

2.9 Web Cache Control Protocol (WCCP)

Dieses von Cisco entwickelte Protokoll setzt nicht mehr allein beim Proxyserver an. Eingehende Webanfragen werden schon in einem WCCP-fähigen Router »abgefangen« und auf die im Router konfigurierten Proxyserver verteilt. Dabei kann bereits im Router anhand der IP-Adresse eine Lastverteilung auf die Proxyserver erfolgen.

2.10 Hypertext Caching Protocol (HTCP)

Dieses Protokoll befindet sich bisher noch im Entwurfsstadium. Es stellt eine Erweiterung von ICP dar und soll dieses zukünftig ersetzen.

Das Prinzip von ICP wird dabei um die Übertragung von Header-Informationen innerhalb der ICP-Pakete erweitert. Dadurch kann ein anfragender Proxy qualifiziertere Entscheidungen treffen, z. B. welcher Nachbar ein aktuelleres Objekt im Cache hat. HTCP soll durch diese Erweiterungen deutlich mehr Ressourcen benötigen, wodurch es bei einigen Tests zu Verzögerungen von Anfragen gekommen sein soll. Bisher sind kaum Produktiveinsätze bekannt.