Anwendung von Content Delivery Networks:

Medien-Distribution

Ali Ikinci

Februar 2005

Seminararbeit

Content Delivery Networks

Lehrstuhl für Praktische Informatik IV

Prof. Dr. W. Effelsberg

Betreuer: Christian Liebig

Universität Mannheim

Inhaltsverzeichnis

1 Einführung

Neue Multimediastandarts und Protokolle, Breitbandanbindung einer großen Anzahl von Benutzern - auch in Schwellenländern - und das veränderte Konsumverhalten der Nutzer, haben ungeahnte Herausforderungen an die Leistungsfähigkeit des Internets gestellt.
Audio- und Video-Codecs gab es auch vorher, jedoch hat die Effizienz und die Verbreitung dieser grosse Ausmaße angenommen [1]. MP3 und DIVX sind Synonyme für ein neues Verständnis des Internets geworden. Flash und Java haben eine erweiterte, virtuelle und interaktive Internet-Welt geschaffen. Das Internet hat einen festen Platz in der Unterhaltung neben dem Fernseher eingenommen, zahlreiche kommerzielle Produkte wie Windows Media Center Edition [2] stellen dies unter Beweis. Die alten, statischen und inhaltsunabhängigen Strukturen wurden durch neue, dynamische, inhaltsbasierte Strukturen ersetzt, welche Content Delivery Networks genannt werden. Ziel dieser ist es, durch den Einsatz verschiedener Techniken, unnötige Netzlast zu vermeiden, Lokalität der Daten zu gewährleisten, um damit die Latenzzeit sowie die Erreichbarkeit von Content-Diensten sicherzustellen. Ziel dieser Arbeit ist es, CDNs, die auf Mediendistribution spezialisiert sind, näher zu betrachten.
Es gibt in diesem Gebiet verschiedene kommerzielle und freie Lösungen, die sich im Laufe der Zeit zu Quasi-Standartds entwickelt haben. In dieser Seminararbeit werde ich einige weit verbreitete und freie, im Sinne der GPL [3], Systeme. Ich habe bei meiner Auswahl freie Systeme bevorzugt, da bei diesen meist detailliertere Dokumentation und besserer Support besteht, zudem wird der Quellcode der Programme mitgeliefert. Ferner wird in dieser Arbeit speziell auf die Implementierung der technischen Gegebenheiten der Systeme eingegangen. Hauptthemengebiete in diesem Seminar waren, Cachingverfahren, Routing in Content Delivery Netzwerken, Peer-2-Peer-Netzwerke und Content-Repurposing.

2 Grundlegende Theorie zum Thema

Iin diesem Kapitel werden grundlegende Verfahren, die zum Verständnis des Themas dienen, näher beschrieben. Neben HTTP, dem Hypertext Transfer Protocol, wird auch auf Caching und Streaming eingegangen.

2.1 HTTP

Das World Wide Web arbeitet mit HTTP. Es wird hauptsächlich für den Datentransfer zwischen Web Servern (z.B. Apache [4]) und Web Clients (z.B. Mozilla Firefox [5]) benutzt. Auch bei vielen

anderen Anwendungen in der ein verbindungsbasierter Datentransfer benötigt wird, kann HTTP eingesetzt werden. Unter anderem auch im Gnutella Netz.
Dem HTTP *request* (Anfrage) liegen drei Hauptbestandteile zu grunde: die Anfragemethode, der Uniform Resource Locator (URL) und eine Reihe von Anforderungs-Headern. Eine HTTP *reply* (Anwort) besteht aus einem numerischen Ergebniscode, einer Reihe von Antwort-Headern und einem optionalen Antwort-Body (Hauptteil). Die am meisten genutzte Anforderungsmethode ist GET, welche eine Anforderung zum Erhalt des Objektes mit der gegeben URL darstellt. Das GET *request* ist ein *download*-Vorgang. Eine weitere oft genutzte Anforderungsmethode ist POST, welcher ein *upload*-Vorgang ist.

2.2 Caching

Caches werden zum zwischenspeichern von Inhalten zwischen zwei Parteien eingesetzt. Sie verbessern die Latenzzeit und entlasten das Netz. Zudem wird der Content-Server geschont. Diese Caches können in einer Hierarchie miteinander vernetzt werden um ihre Effizienz zu erhöhen. Engpässe werden so vermieden und die Lokalität der Benutzer ausgenutzt.

2.3 Streaming

Wenn man Audio- oder Video-Content anbietet, ist es oft so, dass die Benutzer nicht unbedingt am gesamten Inhalt dieser interessiert sind, und/oder dass die Datenmengen zu groß sind, um schnell genug geladen zu werden. Bei Multimediainhalten die in Echtzeit generiert werden, wie bei Internetradios ist es überhaupt nicht möglich, den Audio-Stream vorher herunterzuladen, hier kommt das Streaming zum Einsatz. Es ermöglicht mehreren Benutzern einem einzelnen Audio- oder Video-Stream zu folgen und stellt sicher, dass auch nur soviel Bandbreite genutzt wird, wie aktuell benötigt wird. Es werden immer jeweils die nächsten Frames geladen, die abgespielt werden sollen. Für diese Art der Nutzung von Medienstreams wird als Bandbreite beim Client nur jeweils soviel wie die Bitrate des Audio- oder Videofiles benötigt. Als Beispiel sei die Möglichkeit genannt, einen Trailer eines neuen Kinofilms anzusehen. Der Benutzer kann fast gleichzeitig mit dem Ansehen beginnen. Wenn ihm der Film nicht gefällt bricht er ab und es wird nur ein Teil des Videofiles über das Netz übertragen.

2.4 Peer-to-Peer-Netze

Peer-to-Peer-Netze sind dezentrale, dynamische Netze die über keine Kontoll- oder Authentifikationsinstanzen verfügen. Jeder Peer kann sich mit dem Netz verbinden und zu jederzeit

wieder aus dem Netz austreten. Es gibt für die Eigenschaften des Netzes keine Garantien und Sicherheiten. Sie werden zu einem bestimmten Zweck aufgebaut und erfreuen sich einer großen Beliebtheit. Datenaustausch von geschützen Inhalten und Softwarepiraterie sind leider allzu oft die Absicht der meisten Teilnehmer. Dank dem HTTP-Protokoll, welches in den meisten dieser Netze Einsatz findet, wäre es auch möglich eine neue Art des World Wide Web so zu erzeugen.

3. Andwendungsbeispiele

Im folgenden werden drei Anwendungsbeispiele für die drei Themengebiete – Caching, Streaming und Peer-to-Peer-Netze – behandelt.

3.1 Squid – ein Proxy Server

Normalerweise stellen Web-Clients direkte Verbindungen zu einem Web-Server her. Weiterhin besteht die Möglichkeit, dass der Web-Client so eingestellt wird, dass er sich mit einem Proxy Server in Verbindung setzt und dieser seine Anfragen für ihn weiterleitet. Der Proxy dient dem Client als eine Art Internet-Gateway und kann noch für andere Aufgaben, wie zum Beispiel Content-Filtering mit Access-Control-Lists[9], erweitert werden. In Abbildung 1 ist ein Anwendungsbeispiel für einen transparenten Proxy-Server aufgeführt, welcher automatisch zwischen die Verbindung geschaltet wird.

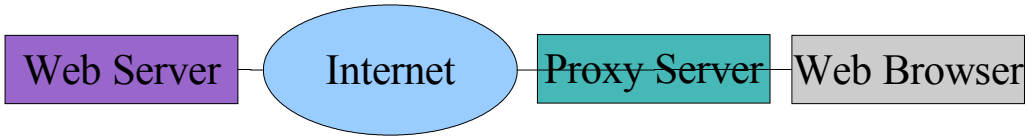

Abbildung 1. Verbindung über einen Proxy Server

Da der Cache-Server die Anfragen lokal zwischenspeichern kann, besteht ein großes Interesse daran, dass dieser auch nur die populärsten Objekte speichert, um nicht unnötig Plattenplatz zu belegen.
Objekte sind jegliche Daten - z.B. Bilder oder Dokumente -, die eine eindeutige URL haben und über HTTP erreichbar sind. Es wird die URL mit dem Anforderungsheader gespeichert.
Ein Cache-Hit bedeutet, dass eine gültige Kopie des angeforderten Objektes im Cache existiert. Ein Cache-Miss dagegen steht für eine im Cache fehlende Anforderung. Der Erfolg eines Caches wird an seiner Cache-Hit-Rate bemessen.
Squid ist das Ergebnis der Bestrebungen von zahlreichen Individiuen der Internet Community. Die Entwicklung wird geleitet von Duane Wessels vom National Laboratory for Applied Network Research und gefördert von der National Science Foundation. Squid ist abgeleitet aus dem Harvest Forschungsprojekt des ARPA.
Squid hält Metadaten und insbesondere *hot objects* im RAM, speichert DNS-lookups und

implementiert negatives Caching von fehlgeschlagenen Anfragen.

3.1.1 Hierarchisches Caching

Cache Hierarchien sind eine logische Weiterführung des Caching-Konzepts. Ein zentraler Cache-Server kann zu einem Engpass werden, wenn zu viele Anforderungen bestehen. In einer Cache-Hierarchie können die Caches nicht nur Lokalität gewährleisten, sondern auch eine viel grössere Anzahl an Anforderungen bearbeiten (Load). [9]
Dafür werden die Cache Server in einer Hierarchie miteinander vernetzt in der einige Cache Server von anderen zwischengespeichert werden. Es gibt Nachbarn (*peers* oder *siblings*) und Eltern (*parents*), und je nach Konfiguration kann man sie variiren, wie es benötigt wird. In Abbildung 2 ist ein Beispiel für eine Konfiguration mit zwei Nachbar- und einem Vatercache aufgeführt.

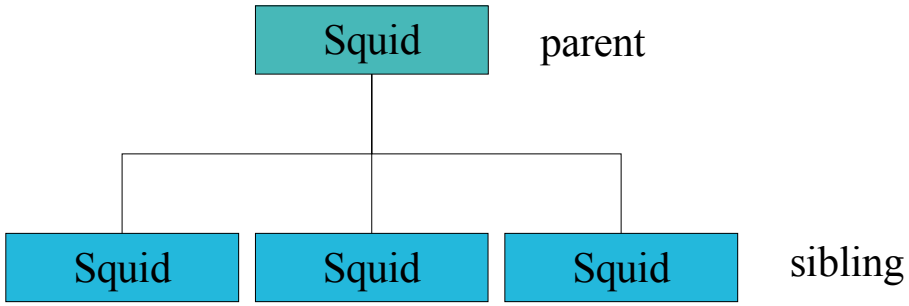

Abbildung 2. Hierarchisches Caching

Die Konfiguration kann jeweils lokal auf dem jeweiligen Cache-Server geschehen. Falls es nur Nachbarn gibt, nennt man diese Konfiguration auch verteiltes Caching.
Squid kann hervorragend mit anderen Proxy-Servern kommunizieren, es unterstützt folgende Inter-Cache-Kommunikationsmechanismen; ICP [10], Cache-Digests [11], HTCP (Hyper-Text Cache Protocol) [12] und das CARP (Cache Array Routing Protocol) [13]. Diese werden benötigt, um eine effiziente Kommunikation in einem Netz bzw. in einer Hierarchie von Cache-Servern zu gewährleisten. [14]
Die Primärfunktion eines Inter-Cache-Protokolls ist es, die Objektduplizierung zu vermeiden und die Trefferquote zu erhöhen. In einem Content Delivery Network mit verteilten Cache-Servern möchte man die Objekte möglichst in jedem Cache-Server speichern, auch wenn ein anderer Cache Server dies bereits gemacht hat. Geringe Netzwerklatenz und Skalierbarkeit werden durch die Ausnutzung der Lokalität der Caches sichergestellt, obwohl damit Speicherkapazität belegt wird. [9]
Die Einstellung von Squid erfolgt hautsächlich über eine zentrale Konfigurationsdatei in *squid.conf*. Dabei gibt der `cache_peer` tag an, mit welchen Cache-Servern Squid in Verbindung stehen soll; z.B.:

```
cache_peer cache.test.org parent 3128 3130 default
```

Hier ist der Cache-Server cache.test.org der Vater und die Kommunikation erfolgt über die Ports 3128 (HTTP Verkehr) und 3130 (ICP Kommunikation). Abbildung 3 zeigt eine solche Verbindung .

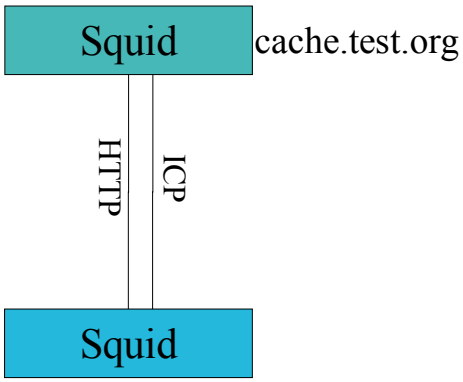

Abbildung 3. Konfiguration mit einem Parent Cache

3.1.2 ICP – Das Inter-Cache-Protokoll

Das Inter-Cache-Protokoll [10] gewährleistet eine schnelle und effiziente Methode der Inter-Cache-Kommunikation mit einem Mechanismus, um komplexe Cache-Hierarchien zu ermöglichen. ICP ermöglicht es einem Squid, einen anderen Cache zu fragen, ob er eine gültige Kopie eines benannten Objektes hält und erhöht somit die Chancen, den Nachbar-Cache auszuwählen, der einen Cache-Hit gewährleisten wird. ICP Anfragen können auch untereinander weitergereicht werden.
Squid benutzt das ICP außerdem, um den aktuellen Zustand des Netzes zu bestimmen, indem es Antwortzeiten (Round-Trip-Time) und fehlende Antworten auswertet. ICP Messages werden als UDP-Pakete übertragen.
Eine Beispielkonfiguration für einen Eltern- und drei Nachbarcaches mit ICP würde wie folgt aussehen:

```
local_domain mydomain.net
cache_host cache.isp1.net parent 3128 3130 no-query default
cache_host cache.isp2.net sibling 3128 3130
cache_host cache.isp3.net sibling 3128 3130
cache_host cache.isp4.net sibling 3128 3130
```

ICP wird nutzlos wenn starke Überlastungen der Verbindung (*congestion*) auftritt, obwohl eigentlich genau hier Caching benötigt würde. Desweiteren fügt ICP einem Request die maximalen Antwortzeiten der Nachbarcaches hinzu, was ICP für verschiedene Anwendungsbereiche, wie Quality of Service, unbenutzbar macht. Hier erweist sich das neuere Cache-Digest [11] als Lösung dieses Problems. [15]

3.1.3 Cache-Digests

Cache-Digests (Zusammenfassungen) sind eine Antwort auf die Probleme der Latenz und der Überlastung im Zusammenhang mit vorherigem Inter-Cache-Protokolle.
Squid hält eine Hash-Liste seiner Objekte im Hauptspeicher. Diese auf MD5 [16] basierende Liste hilft Squid herauszufinden, ob ein Objekt im Cache ist, ohne zu große Mengen Hauptspeicher zu belegen oder Plattenzugriffe zu benötigen. In gewissen Abständen speichert Squid diese Liste in einer kleinen Bitmap ab. Wenn ein Bit aktiviert ist, bedeutet es, dass dieses Objekt im Cache ist, sonst nicht. Diese Zusammenfassung ist für andere Caches über HTTP mit einer speziellen URL verfügbar. Falls ein Cache feststellen will, ob sein Nachbar ein Objekt hält, welches dieser benötigt, führt er die gleiche Hash-Funktion aus, die die Werte in der Bitmap erzeugt hatte. Falls der Cache dieses Objekt hält, muss dieses Bit aktiviert sein. [9] Es besteht die Möglichkeit des Cache-Miss, falls der Eintrag in dem Digest bereits veraltet ist. Dieses trifft aber selten ein und wird dann einfach direkt von der Quelle angefordert.

Public Keys

Die Schlüssel, nach denen in Cache Digests gesehen wird, werden von der MD5 Funktion aus der Verkettung von:

1. einem Wert für die benutzte HTTP request Methode und
2. der angefragten URL

berechnet.
Gültige Werte für die HTTP Methode sind:

GET	1
POST	2
PUT	3
HEAD	4
CONNECT	5
TRACE	6
PURGE	7

Der Methodenwert wird als 8-Bit *integer* gespeichert.
Als Beispiel sei der Public Key für "http://www.w3.org" angefragt mit der HTTP "GET" Methode:

e06a56257d8879d9e968e83f2ded3df7

berechnet aus:

```
ASCII:  GET  h  t  t  p  :  /  /  w  w  w  .  w  3  .  o  r  g  /
Hex:    01   68 74 74 70 3a 2f 2f 77 77 77 2e 77 33 2e 6f 72 67 2f
```

Die Cache Digest Hash-Funktion [11]

Die Anzahl der Bits pro Public Key ist in Squid standartmässig auf 5 gesetzt. Die Größe des Digest, welche als Kapazität bezeichnet wird, hängt ab von den Bits pro Public Key und der Anzahl an Public Keys, die im Digest gespeichert werden können. Sie lässt sich folgendermaßen berechnen:

*digest_size=int((capacity*bits_per_entry+7)/8)*

Diese Werte müssen an alle Caches weitergeleitet werden, die von diesem Digest Gebrauch machen wollen. Dies bewerkstelligt der Cache-Digest-Header.

Um einen gegebenen Public Key in einem Cache-Digest einzutragen, wird eine Anzahl von Indizes in dem Array, mit einer Hash-Funktion, berechnet (in Squid N=4). Dazu werden folgende Schritte ausgeführt

1. Berechnung des Public Key aus der HTTP Methode und der URL
2. Teilung des resultierenden 128-bit-Wert in N Teile:
   key[0], ... , key[N-1]
3. Für jeden dieser N-Teile ist der korrespondierende Index im Cache Digest der Digestwert Modulo der Anzahl der Bits im Digest.
   hash_key[i]=key[i] % (digest_size*8)

Die resultierenden N Indizes sind das Ergebnis der Hash-Funktion. N wird die Dimension der Hash-Funktion genannt.

Im obigen 'GET http://www.w3.org/' Beispiel, ist der Public Key:

e06a56257d8879d9e968e83f2ded3df7

Im aktuellen Squid Schema (N = 4) würde das in vier Teile aufgeteilt werden:

```
key[0] = 0xe06a5625;
key[1] = 0x7d8879d9;
key[2] = 0xe968e83f;
key[3] = 0x2ded3df7;
```

Und die Indizes werden folgendermaßen berechnet:

```
hash_key[0] = key[0] % (digest_size * 8);
hash_key[1] = key[1] % (digest_size * 8);
hash_key[2] = key[2] % (digest_size * 8);
hash_key[3] = key[3] % (digest_size * 8);
```

Dies ergibt folgende Indizes:

```
hash_keys[0] = 0x05;
hash_keys[1] = 0x29;
hash_keys[2] = 0x5f;
hash_keys[3] = 0x17;
```

Das sind die Indizes in dem Bit-Array.

Wie zu sehen war, ist Squid in der Lage, Cache-Hierarchien aufzubauen, ihre Kommunikation

untereinander erfolgt mit den vorgestellten Protokollen. Das ist die Grundlage für das Auslagern des Content auf den “Rand” des Internets oder die Lastverteilung eines Content-Servers beim Content-Anbieter.

3.2 Icecast2 und IceS – Ein Streaming-Media-Server-System

Es gibt zwei Bestandteile für die meisten Streaming Media Server: die Komponente, die den Inhalt zur Verfügung stellt (der Source Client) und die Komponente, welche dafür zuständig ist, den Inhalt den Benutzern bereitzustellen (der Streaming Server). Mit diesem Layout kann der Source-Client auf einem anderen, entfernten System als der Streaming Server sein. Ein Client für diesen Streaming-Media-Server muss dann in der Lage sein, einen solchen Inhalt zu empfangen, wie z.B. WinAmp oder XMMS für MP3 und Ogg.
Icecast2[18] ist ein Streaming Media Server der Ogg Vorbis, AAC, MP3 Audiostreams und Theora Videostreams unterstützt. Es kann dafür benutzt werden, um ein Internet-Radio, Internet-TV, eine private Jukebox oder ähnliches aufzusetzten. Icecast2 ist sehr vielseitig und unterstützt offene Standarts für die Kommunikation und die Interaktion. Es bietet die Möglichkeit des Cachens von anderen Streaming-Servern und das Clustering für einen Streaming-Server. Icecast2 findet eine große Verwendung im Gebiet des Internet-Radios, so sind allein im Icecasts YP-Verzeichnis [19] über 5000 Sendungen verzeichnet.
IceS[20] ist ein Source Client für einen Streaming-Media Server. IceS sendet den Media-Stream (codierte Daten über ein Socket) an Icecast, der diesen dann an die Clients weiterleitet. [21]

3.2.1 Aufbau

Jeder Icecast-Server kann mehrere Sendungen (oder Mountpunkte) haben die verschiedene Medienstreams beinhalten. Ein Client kann zur gleichen Zeit nur jeweils einer Sendung beitreten. Die Sendungen des Servers können inhaltlich verschiedene Sendungen sein, oder auch Sendungen mit gleichem Inhalt jedoch mit verschiedenen Bitraten, was dann einem Content Repurposing auf der Serverseite entsprechen würde.[22]
Icecast wird über die XML Konfigurationsdatei *icecast.xml* eingestellt. Dort findet man unter anderem auch Einstellungen für den Port an dem Icecast2 auf Verbindungen wartet und das Passwort für den Source-Client. Folgender Aufruf startet dann den Server:

```
icecast -c /path/to/icecast.xml
```

Danach wird IceS mit dem Passwort und dem Port mit Icecast verbunden.
Der Client muss dann nur noch *http://yourip:port/mounpointyouspecified.m3u* mit seinem Browser oder direkt mit seinem Media Player aufrufen. Diese Playliste enthält dann normalerweise alle

Mountpunkte, die man dann selbst auswählen kann.

Für die Skalierbarkeit des System ist es dann wichtig in der Konfigurationsdatei verschiedene Einstellungen anzupassen.

Die maximale Anzahl der Clients wird in der Einstellung `clients` festgesetzt, jeder weitere Verbindungsversuch wird abgelehnt.

Die Einstellung `burst-on-connect` bewirkt einen *burst* bei der ersten Verbindung eines Clients. Somit wird erreicht, dass die Startzeit des Clients verkürzt wird, da die meisten Media Player lokale Puffer haben die erst aufgefüllt werden müssen, bevor der Stream starten kann. Das erzeugt eine Latenz zwischen Sender und Empfänger, welche mit *burst-on-connect* 3 Sekunden und ohne *burst-on-connect* 1,5 Sekunden beträgt. [22] Abbildung 4 zeigt eine Beispielkonfiguration zwei Clients und einem Source-Client.

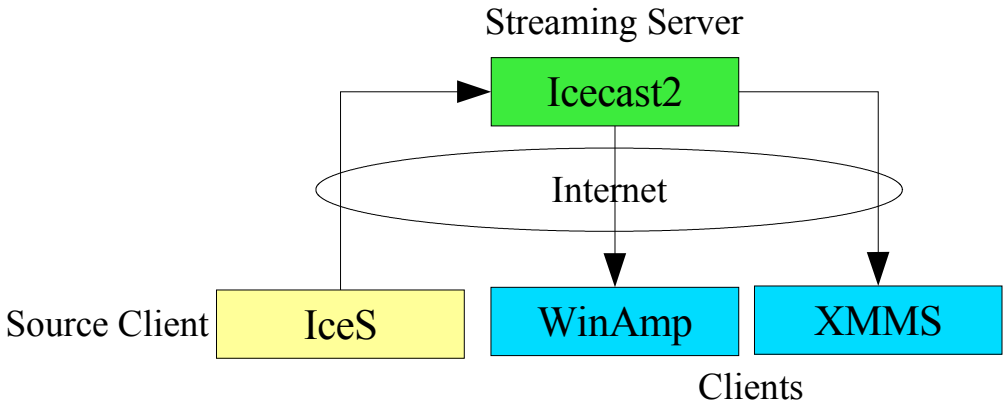

Abbildung 4. Ein Streaming-Server-System mit Icecast2 und IceS

3.2.2 Relaying

Relaying ist der Vorgang bei dem ein Streaming Server einen oder mehrere Streams eines entfernten Servers cacht. Diese Option wird eingesetzt wenn der Streaming Server bei Enpässen wie Netzlast oder Systemlast nicht alle Clients zur gleichen Zeit bedienen kann.

Der Relay spiegelt alle Mountpunkte des Master-Servers und prüft in einem bestimmten Intervall den Master-Server auf eventuelle Änderungen. Abbildung 5 zeigt ein Beispiel mit der nachfolgenden Konfiguration. Es werden dafür in der Konfiguration des Mirrors folgende Einstellungen angepasst

```
<master-server>icecast-server.dyndns.org</master-server>
<master-server-port>8001</master-server-port>
<master-update-interval>120</master-update-interval>
<master-password>secret</master-password>
```

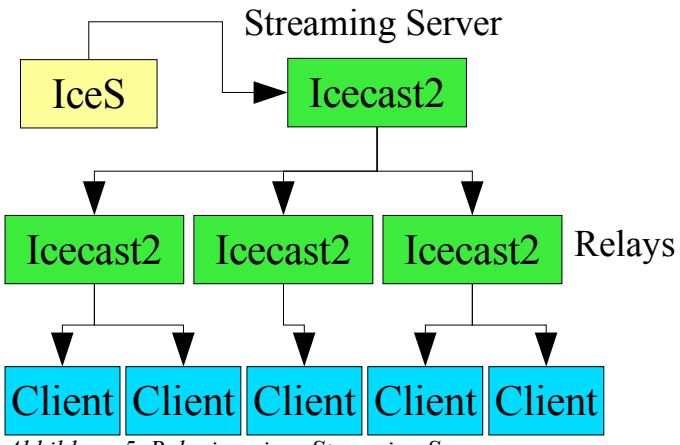

Abbildung 5. Relaying eines Streaming Servers

3.2.3 Clustering

Beim Clustering werden mehrere Relays, nach einer Authentifikation durch ein Passwort, miteinander in einem Stream Directory gelistet. Es können nur gleiche Mountpunkte oder gleiche Server geclustert werden. Der Stream, der geclustert wird, kommt nur einmal in der Stream-Directory vor. Der Benutzer erhält eine m3u Playlist, in der die verschiedenen Cluster für diesen Stream aufgeführt werden. Mit dieser Option ist es möglich, Lokalitäten auszunutzen, wenn man Cluster anwählen kann, die in der Nähe sind, was allerdings noch manuell erfolgen muss.

3.2.4 Yellow Pages

Ein YP-Verzeichnis ist eine Liste von aktuellen Streams. Icecast hat sein eigenes YP-Verzeichnis bei http://dir.xiph.org. In einer YP kann man dann verschiedene Informationen zu den Streams wie Genre Bitrate, aktueller Titel etc. erfahren und sich dann auch gleich verbinden lassen. Auf Wunsch verbindet sich Icecast2 selbst mit dem YP Verzeichnis und aktualisiert die Daten regelmäßig.

3.3 Gtk-Gnutella – ein Gnutella-Net Servent

Das Gnutella Netzwerk ist ein reines Peer-to-Peer Netzwerk, alle Hosts werden gleich behandelt ohne Rücksicht auf ihre Bandbreite, CPU Leistung oder andere Eigenschaften. Die Kommunikation der Knoten untereinander erfolgt über das Gnutella Protokoll [24], der Datentransfer über das HTTP-Protokoll [25]. Neue Hosts können sich jederzeit mit dem Netzwerk verbinden und vorhandene das Netzwerk verlassen. Jede Gnutella Software ist gleichzeitig beides Server und Client, da sie bidirektionalen Datentransfer gestattet. Die Gnutella Entwickler nennen diese Software “servent”. Das Gnutella Protokoll ist eine Art von Aufruf-Antwort (call-and-response) Protokoll, welche komplizierter ist als nur eine Weiterleitung von News oder E-Mail, wie z.B. bei UUCP. Abbildung 6 zeigt die Funktionsweise des Protokolls. Angenommen A frägt B nach Daten, die zu “MP3” passen.

Nachdem B alles, was zur Anfrage passen könnte, zurückgibt, leitet er die Anfrage an seinen Kollegen C weiter. Aber anders als bei E-Mail oder News protokolliert B die Anfrage von A. Falls C etwas passendes hat, sendet es diese Information an B, welches noch weiß, dass die Anfrage von A kam.

Abbildung 6 Kommunikation im Gnutella-Netz

Dieser Abschnitt beschäftigt sich mit den Eigenschaften von Gtk-Gnutella [26] einer freien Gnutella Servent Software für Unix-Derivate.

3.3.1 Caching – Das Gnutella Web-Caching-System

Das Ziel des Gnutella Web-Caching-System[27] (der "Cache") ist das Eliminieren des "Initial Connection Point Problem", eines vollständig dezentralen Netzwerks. "Wo finde ich einen ersten Host für meine Verbindung?". Der Cache ist ein Skript, das auf einem Web Server läuft, der IP Adressen von Gnutella Hosts und URLs von anderen Caches enthält. Gtk-Gnutella verbindet sich mit einem zufälligen Cache in seiner Liste. Er sendet und empfängt dann IPAdressen und URLs vom Cache. Mit der zufälligen Verbindung soll sichergestellt werden, dass alle Caches eventuell voneinander erfahren und dass sie relativ aktuelle Hosts und URLs speichern. Dieses Konzept ist unabhängig von den Gnutella Servents.

3.3.2 Query Routing für das Gnutella Netzwerk

Das Gnutella Protokoll stellt Suchanfragen per Broadcast. Diese Vorgehensweise kann nicht erwünscht sein da das Netz exponentiell wachsende Suchanfragen zu bewältigen haben müsste. Man muss also versuchen Suchanfragen nur an Servents zu schicken, die gesuchte Inhalte auch wirklich liefern können. Die Idee des Query-Routings[28] macht sich diese Tatsache zu Nutze und erstellt Routingtabellen auf der Basis von Schlüsselwörtern.
D.h., es werden alle Schlüsselwörter, nach denen auf dem aktuellen Host gesucht werden kann, in einer Liste zusammengefasst, und an die Nachbarn verschickt. Die Tabellen enthalten auch

Distanzinformationen, weshalb die Hashwerte nicht eine Bitmap sind, sondern ein Array von Zahlen mit Anzahl der Hops.

Als Beispiel sei folgendes gegeben, wobei das Netz nicht in Baumform sein muss und auch Zyklen beinhalten kann. Nach einem Zeitschritt haben die Hosts ihre Routingtabellen einen Hop weiter geschickt. Das kann man auf der linken Seite beobachten. Es werden hier Schlüsselwörter angezeigt anstatt gehashte Werte. Die Tabelle in Abbildung 7 "{bad/1, joke/2}" würde z.B. wirklich als das Feld [∞, 2, ∞, 1, ∞, ...] existieren, falls "bad" zu 3 gehasht würde und "joke" zu 1. Nach einem zweiten Zeitschritt, würden die Hosts die Routingtabellen zwei Hops weiter geschickt haben, was man auf der rechten Seite beobachten kann. Man kann jetzt sehen, dass A die Routingeinträge für alle Dateien hat und B gemerkt hat, dass A keine Dateien hat.

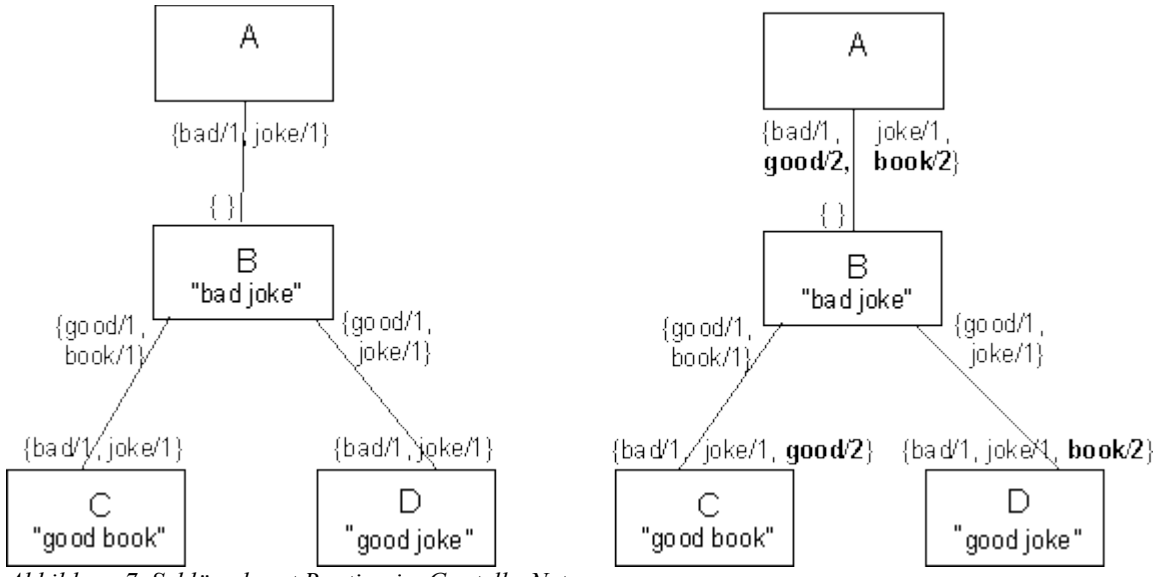

Abbildung 7. Schlüsselwort Routing im Gnutella-Netz

3.3.3 Ultrapeers

Trotz all den guten Features die Gnutella hat und dem Query-Routing, wurden immer wieder Zweifel über die Skalierbarkeit des Gnutella-Netzes laut. Hauptsächlich wegen der Fülle von Messages, die durch das Gnutella Netz fliessen, darunter auch Broadcast Pings und Queries wie auch die Unfähigkeit, die Content-Verfügbarkeit nah genug zum suchenden Host zu bringen. Das Schema der Ultrapeers [29] geht einen Schritt weiter und implementiert eine Zwei-Level-Hierarchie von Knoten im Gnutella Netz, die schneller sind (Bandbreite und CPU power), und den anderen. Diese schnelleren Knoten werden zu Ultrapeers und nehmen mehr Last auf sich als die normalen Knoten (die Blattknoten). Ultrapeers halten viele (10-100) Verbindungen zu Blattknoten und einige (<10) Verbindungen zu anderen Ultrapeers. Diese Ultrapeers schirmen die Blattknoten fast vom ganzen Ping und Query-Verkehr ab. Abbildung 8 zeigt eine mögliche Konstellation eines Ultrapeers. Das

geschieht dann auf der einen Seite durch QRP Routing und auf der anderen Seite dem Reflector-Indexing, bei dem die Blattknoten alle zur Verfügung gestellten Dateien dem Ultrapeer melden. Dieser Mechanismus funktioniert auch sehr gut mit alten Servents, da diese von der Existenz des Ultrapeers nichts mitbekommen.

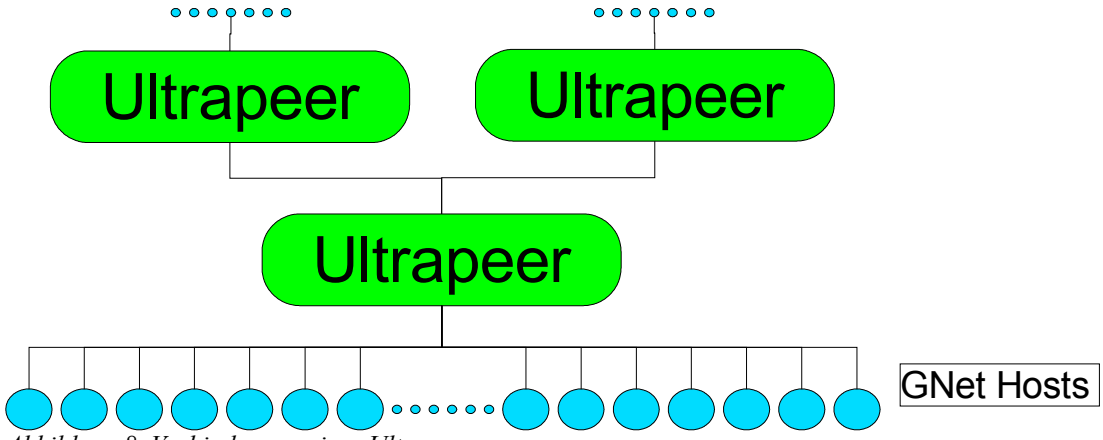

Abbildung 8. Verbindungen eines Ultrapeers

4 Zusammenfassung und Ausblick

In der Informationstechnologie sind Konzepte und Ideen immer nur so gut, wie sie Anwendung finden. Mit einem guten theoretischen Fundament und einer aktuellen Kenntnis der Anwendungen ist es dem modernen Informatiker möglich, sich für die richtige Lösung seines Problems zu entscheiden. Mit diesem kleinen Einblick in die Anwendungen von Content-Delivery-Networks war zu sehen, wie einige der Konzepte, die in den vorhergehenden Seminaren vorgestellt wurden, im der Realität Anwendung finden.

Synergieeffekte sind dadurch entstanden, dass die verschiedenen Konzepte jeweils auf eine besondere Art und Weise miteinander verknüpft sind, das jeweilige Problem am besten zu lösen. Genau dieses Gespühr gilt es in einer dynamischen Netzwerkwelt der Zukunft zu schärfen, um neue Herausforderungen mit alten Ideen und einem neuen Verständnis zu meistern.

5 Literaturverzeichnis

[1] François Pachet - CONTENT MANAGEMENT for Electronic Music Distribution, COMMUNICATIONS OF THE ACM April 2003/Vol. 46, No. 4

[2] Windows Media Center Edition http://www.microsoft.com/windowsxp/mediacenter

[3] Free Software Foundation: Gnu General Public License http://www.gnu.org/licenses/gpl.html

[4] Apache Http Server Project http://httpd.apache.org/

[5] Mozilla Firefox Web-Browser http://www.mozilla.org/products/firefox/

[6] Squid http://www.squid-cache.org/

[7] Macromedia Flash http://www.macromedia.com/software/flash/

[8] Sun Java Technology http://java.sun.com/

[9] Squid Handbook http://squid-docs.sourceforge.net/latest/book-full.html

[10] ICP – RFC 2186 http://www.squid-cache.org/Doc/rfc2186.txt

[11] CacheDigest-v5-spec http://www.squid-cache.org/CacheDigest/cache-digest-v5.txt

[12] HTCP – RFC 2756 http://www.faqs.org/rfcs/rfc2756.html

[13] CARP - Cache Array Routing Protocol v1.0 – V. Valloppillil - Microsoft Corporation, K.W. Ross - University of Pennsylvania, 26 Feb 1998 http://www.welltall.com/ymc/WebCaching/in-drafts/draft-vinod-carp-v1-03.txt

[14] Squid FAQ http://www.squid-cache.org/Doc/FAQ/FAQ.html

[15] Configuring Hierarchical Squid Caches http://www.squid-cache.org/Doc/Hierarchy-Tutorial/

[16] R. Rivest - The MD5 Message-Digest Algorithm, MIT Laboratory for Computer Science and RSA Data Security, Inc. April 1992 http://www.faqs.org/rfcs/rfc1321.html

[17] Squid Programmers Guide http://www.squid-cache.org/Doc/Prog-Guide/

[18] Icecast2 http://www.icecast.org/

[19] Icecast2 Stream Directory http://dir.xiph.org/index.php

[20] IceS Source Client http://www.icecast.org/ices.php

[21] IceS Documentation http://www.icecast.org/docs/ices-2.0.0/

[22] Icecast2 Documentation http://www.icecast.org/docs/icecast-2.2.0/

[23] Icecast2 FAQ http://www.icecast.org/faq.php

[24] Greg Bildson, Christopher Rohrs, An Extensible Handshaking Protocol for the Gnutella Network http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/gtk-gnutella/gtk-gnutella-current/doc/gnutella/handshaking_0.6?rev=HEAD&content-type=text/vnd.viewcvs-markup

[25]HTTP v1.1 Specification http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/gtk-gnutella/gtk-gnutella-current/doc/http/ http-1.1.txt?rev=HEAD&content-type=text/vnd.viewcvs-markup

[25] Gtk-Gnutella http://gtk-gnutella.sourceforge.net/

[27] Hauke Dämpfling, Gnutella Web Caching System http://www.zero-g.net/gwebcache/

[28] Query Routing Protocol http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/gtk-gnutella/gtk-gnutella-current/doc/gnutella/QRP?rev=HEAD&content-type=text/vnd.viewcvs-markup

[29] Anurag Singla, Christopher Rohrs, Ultrapeers: Another Step Towards Gnutella Scalability

http://www.limewire.com/developer/Ultrapeers.html