Datum: 09.12.2003

Dieses Dokument: http://studium.baldauf.org/informatik/2003.WS/schwerpunkt/Problemseminar peer2peer data management/Vortrag/Skript.pdf
Vortrag zum Skript: http://studium.baldauf.org/informatik/2003.WS/schwerpunkt/Problemseminar peer2peer data management/Vortrag/Vortrag.pdf

Universität Leipzig
Institut für Informatik

Problemseminar:

peer-to-peer data management

Thema 2:

Filesharing Systeme

Skript zum Vortrag von Xuân Baldauf <xuan--peer2peer-data-management@studium.baldauf.org>



Dieses Skript und der Vortrag wurde betreut von Nick Golovin <golovin@informatik.uni-leipzig.de>

Inhalt

1. Einleitung

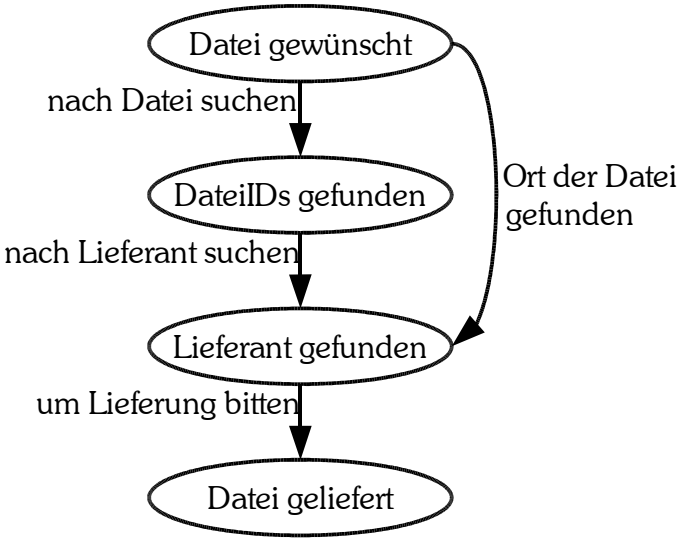
Filesharing ist das Vervielfältigen von Dateien zwischen verschiedenen Nutzern.

Betrachtet man die philosophische Teilung zwischen Materie und Geist, so sind Dateien zu den Informationen und damit zur Domäne des Geistes zuzurechnen. Bei Materie gilt, dass die Nutzung durch einen anderen (B) immer die Nutzung durch den einen (A) beeinträchtigt. Hat beispielsweise A einen Apfel, und wird dieser von B gegessen, dann kann A diesen nicht noch einmal essen.
Dies ist bei Informationen grundsätzlich anders. Nutzt A Informationen, und erhält B diese Informationen, so wird A dadurch grundsätzlich nicht beeinträchtigt. Die Vervielfältigung von Information steigert also grundsätzlich ihren Nutzen mit der Anzahl der Beteiligten, während dies bei Weitergabe von Materie nicht der Fall ist.

Dies ist die Ur-Motivation von Filesharing. Jedoch ist Filesharing spezifischer, denn da geht es nur um Dateien, nicht um Informationen im allgemeinen. Außerdem assoziiert man in der heutigen vernetzten Welt mit Filesharing Dateitransport über Netzwerke. Filesharing im engeren Sinne ist demnach ist das Vervielfältigen von Dateien zwischen verschiedenen Nutzern über ein Computer-Netzwerk.

Funktionsprinzip

Die meisten konkreten Filesharing-Programme und -Netzwerke haben eine ähnliche Vorgehensweise zur Besorgung von Dateien nach folgendem Muster:



Anwendungsfall: Datei besorgen

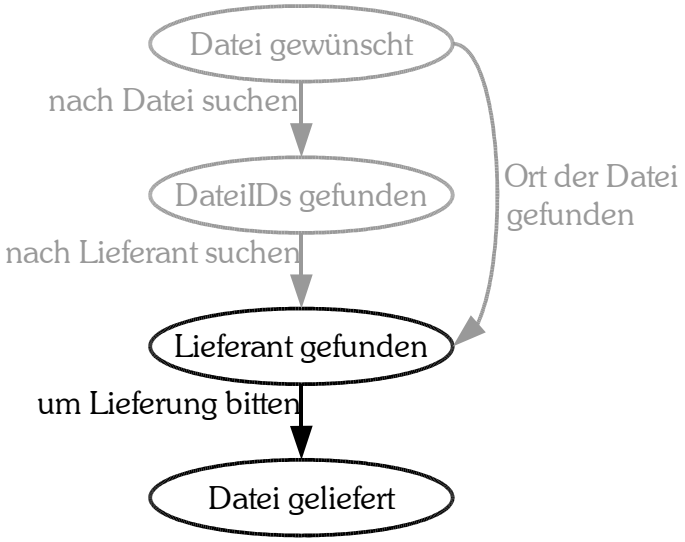
1. Am Anfang steht der Wunsch, sich eine bestimmte Datei zu besorgen. Diese lässt sich oft gar nicht so leicht beschreiben. Aus diesem Grund gibt der Nutzer Beschreibungen ein, die das Programm versteht (etwa ein Wort, was im Dateinamen vorkommen muss, und eine Mindestgröße). Die Software schickt dann eine entsprechende Such-Anfrage in das Netzwerk und präsentiert dem Nutzer eine Liste von darauf passenden Dateien. Jede dieser Dateien war zumindest einmal im Netzwerk verfügbar und hat eine Identifikations-Nummer. Inhaltlich gleiche Dateien haben gleiche Identifikations-Nummern.
2. Glaubt der Nutzer, dass ein Listen-Eintrag einer gesuchten Datei entspricht, so wählt er ihn aus und „bestellt“ die entsprechende Datei zum Download. Die Software sucht nun im Netzwerk anhand der DateiID nach Lieferanten, also Teilnehmern, die diese Datei auch tatsächlich liefern können.
3. Mit diesen wird dann Kontakt aufgenommen. Sie werden gebeten, die Datei zu schicken.
4. Die Datei ist geliefert.

2. Geschichte des Filesharing

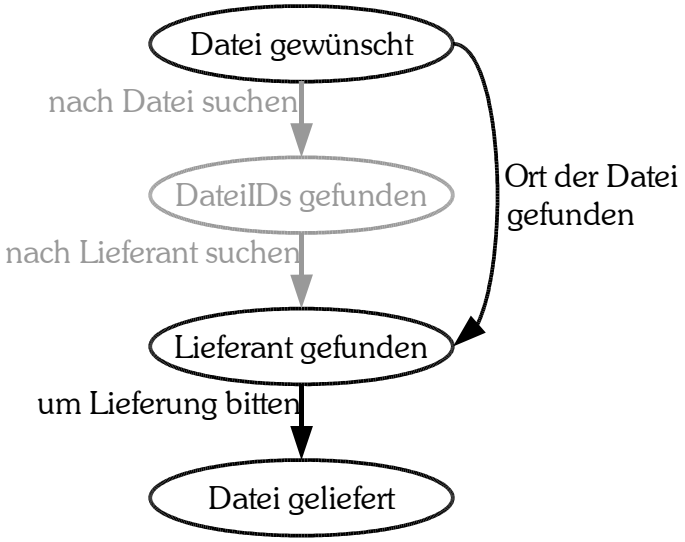
Das Filesharing ist nicht viel jünger als die Datei sowie die Existenz verschiedener Nutzer. Bereits Ende der 1980er Jahre wurden Disketten (mit Software) zwischen den Nutzern ausgetauscht, was dann zu einigen Viren-Epidemien führen sollte. Das Kopieren von Musik-Kassetten war damals schon bekannt. Es ist dokumentiert[1], dass schon im 13. Jahrhundert Bücher in eigens dafür eingerichteten Werkstätten von Hand abgeschrieben wurde.
Diesen Vorgang schnell und Computer-unterstützt durchzuführen ist erst mit dem Aufkommen des (Computer-)Netzwerks möglich geworden. Verbreitete Anwendung hat das Filesharing im engeren Sinne erst durch das Internet gewonnen.

FTP

Bereits seit 1985 ist das File Transfer Protocol[2] spezifiziert. Es erlaubt den Dateitransfer zwischen zwei Rechnern, wobei einer regelmäßig der Server, der andere der Client ist. Mitte der 1990er Jahre entstanden so große öffentliche FTP-Archive[3], die hauptsächlich frei kopierbare Software enthielten. Die Archive spiegelten sich oft gegenseitig, sodass man bei fast jedem FTP-Archiv ähnliche Datei-Kollektionen vorfinden konnte. Nutzer bevorzugten daher den jeweils für sie schnellsten FTP-Server, was oft auch der nach Datenverkehrskosten günstigste war. Rechenzentren konnten somit durch den Aufbau eines eigenen FTP-Servers die Datenverkehrskosten nach außen reduzieren, weil lokale Nutzer tendenziell den lokalen FTP-Server benutzten. Lokale FTP-Server wurden gewissermaßen als vom Nutzer manuell zu durchsuchender, vom Administrator manuell zu konfigurierender Cache für entfernte FTP-Server verwendet. Mit dem Aufkommen von FTP-Such-Servern (welche fast ausnahmslos alle vom Archie[4]-Typ waren) wurde der Aufwand zum Suchen von FTP-Servern für gegebene Dateien stark reduziert.



Anwendungsfall: Datei besorgen bei FTP



Anwendungsfall: Datei besorgen bei FTP mit Archie

1 http://wwwrz.rz.uni-karlsruhe.de/~ea18/mediaevistik/sprachwiss_02/Sprachwissenschaft%2019.5.2002.pdf
2 http://www.w3.org/Protocols/rfc959/
3 zum Beispiel: ftp://ftp.uni-erlangen.de/pub/
4 inzwischen gibt es keine einzige FTP-Suchmaschine des Archie-Typs mehr: http://elfikom.physik.uni-oldenburg.de/Docs/net-serv/archie-gate.html

private FTP-Server

Mit dem Aufkommen des Internet-Zugangs zum Pauschaltarif Ende der 1990er Jahre war es Privatleuten zunehmend finanziell erlaubt, selbst FTP-Server zu betreiben. Da sie selbst schon End-Nutzer waren, machte es wenig Sinn, ihren FTP-Server selbst mit frei kopierbarer Software für die eigenen lokalen Kunden zu füllen. Interessanter waren jedoch Dateien, die öffentlich nicht ohne weiteres frei erhältlich waren: kommerzielle Software und kommerzielle Musik. Inzwischen gab es CD-Laufwerke, mit denen man den Inhalt von Audio-CDs lesen konnte und hochqualitative Audio-Komprimierverfahren[5], die solche Datenmengen fassbar und transportabel machten. So bestand der Sinn vieler privater FTP-Server darin, den Nutzern eine Plattform für Dateiaustausch zu geben. Sie konnten also Dateien herunterladen und selbst Dateien hochladen. Der Nebeneffekt für den Betreiber des FTP-Servers war, dass sich der FTP-Server selbst mit interessanten Dateien füllte. Dieses Konzept wurde jedoch ziemlich schnell von den *Leechern* (wörtlich: „Blutsauger") torpediert. Diese luden jede Menge Daten vom FTP-Server, ohne welche zu ihm zu schicken. Das Resultat war ein starker Verlust der Bandbreite. Die Betreiber reagierten unterschiedlich. Teilweise vergaben sie Accounts[6] nur noch auf Anfrage, statt den Server öffentlich zugänglich zu lassen, teilweise forcierten sie ein Upload-Download-*Ratio*, teilweise beides. Ein U/L-*Ratio* (zum Beispiel 1:3) bedeutete, dass der Nutzer erst dann eine bestimmte Menge Daten (zum Beispiel 6MB) herunterladen durfte, wenn er vorher eine bestimmte Menge Daten (zum Beispiel 2MB) hochgeladen hatte. Dies funktionierte recht gut, die Archive kommerzieller Dateien füllten sich. (Man könnte meinen, dass man statt einer Musik-Datei auch einfach eine Datei mit lauter Nullen hätte hochladen können. Ja, dies ist richtig, wurde aber praktisch kaum angewendet, denn

- Musik-Dateien lagen in der Regel einfacher vor als Dateien mit lauter Nullen (unter Windows war es richtig kompliziert, solche Dateien zu generieren).
- Copyright spielte damals für die Nutzer praktisch noch keine Rolle.
- FTP-Server-Betreiber waren oft sehr schnell im Begutachten frisch hochgeladener Dateien. Missfiel ihnen der Inhalt, so wurde die Verbindung mit dem Nutzer getrennt, der Aufwand war damit umsonst.)

Der Archiv-Aufbau funktionierte so gut, dass teilweise Kommerzialisierungs-Versuche eintraten. Als gerade Werbung im World Wide Web im Trend lag, sollten User auf Werbe-Banner klicken und ein Passwort aus dem Text der Web-Seite zusammenpuzzeln, die nach dem Klick angezeigt wurde, um *Leech*-Zugang zu einem Archiv zu erlangen. Die FTP-Server-Betreiber profitierten von jedem Klick. Mit dem drastischen Verfall der Werbe-Preise im Internet verlor jedoch auch diese Einnahme-Quelle an Bedeutung.

FTP-Server mit kommerziellen Dateien waren oft nicht in einem FTP-Suchmaschinen-Katalog vertreten. Es wurden sich zwar speziell Suchmaschinen für FTP-Server mit „interessantem" Inhalt entwickelt, aber diese waren letztendlich kaum nutzbar, da jeder FTP-Server seine individuellen Restriktionen hatte und oftmals überfüllt war.

Aus diesem Grund war das Filesharing mittels FTP-Server im Wesentlichen ein „Anbieter-Markt": Hatte man einmal Zugang zu einem FTP-Server erlangt, dann graste man dessen Verzeichnisse ab auf der Suche nach etwas Interessantem.

Der Umschwung zum „Nachfrager-Markt" (dass der Nutzer spezifiziert, was ihn interessiert, statt der Server das spezifiziert, woraus der Nutzer wählen kann) kam mit Napster und den

5 also zum Beispiel MPEG Audio Layer 3: http://www.iis.fraunhofer.de/amm/techinf/layer3/

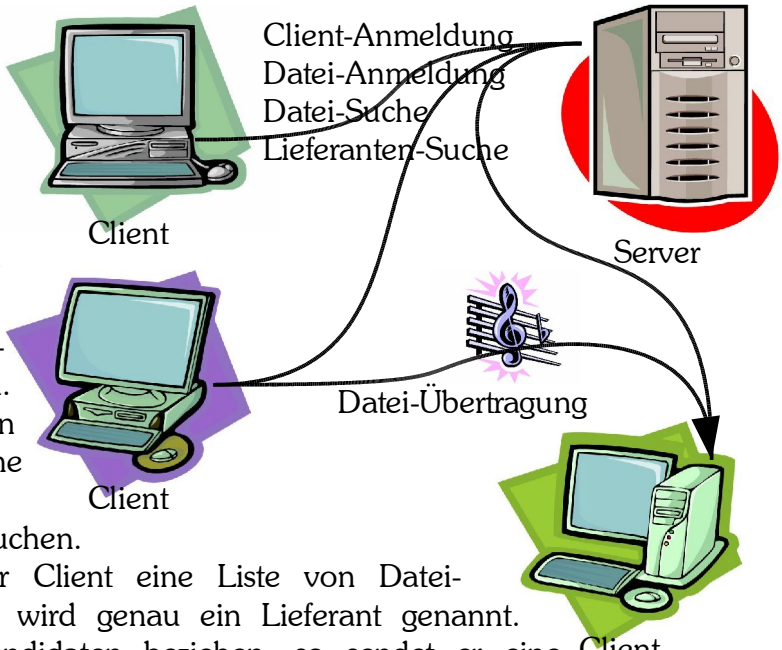
6 also Zugangskennungen mit Name und Passwort

darauf folgenden Filesharing-Systemen. Diese sollten anschließend die privaten FTP-Server letztendlich bis in die Bedeutungslosigkeit verdrängen:

3. Napster

Die erste Software, die gegenseitiges Auffinden, Datei-Suche und Dateiübertragung für den Nutzer vereinte, war Napster. Sie wurde am 1. Juni 1999 vom damals 19-jährigen Shawn Fanning zum Download freigegeben. Napster erleichterte das Filesharing so weit, dass es an Popularität sprunghaft zunahm. Mehrheitlich wird Napster dem Filesharing, nicht jedoch den Peer-to-peer-Netzen zugerechnet, weil Napster einer Client-Server-Architektur unterliegt, wobei nicht jeder Peer Server werden kann.

Architektur



Das Napster-Netzwerk bestand im Wesentlichen aus den Clients und einem Server. (Später gab es auch Unterstützung für mehrere Server, jedoch unterstanden sie alle der Kontrolle einer einzigen Firma.)
Ein Client meldet sich mittels Nicknamen und Passwort beim Server an. Anschließend informiert er den Server über eigene freigegebene Dateien.
Nun kann der Client nach Dateien suchen.
Für die Such-Anfrage erhält der Client eine Liste von Datei-Kandidaten. Zu jedem Kandidaten wird genau ein Lieferant genannt. Möchte der Client einen Datei-Kandidaten beziehen, so sendet er eine Anfrage an den Server, dass er von Lieferant X die Datei mit dem Namen Y beziehen möchte. Dieser leitet diese Anfrage an den entsprechenden Lieferanten weiter. Dessen Zustimmung wird an den Bezieher über den Server geleitet. Anschließend wird eine direkte Verbindung zwischen Bezieher und Lieferant aufgebaut.

Besonderheiten

Allgemein kann man sagen, dass Napster viele Design-Fehler hat, die ohne Mühe vermeidbar gewesen sind. Hier seien einige Besonderheiten erwähnt:

- Napster unterstützt „maskierte[7] Clients“, das heißt Clients, die nur ausgehende, aber keine eingehenden TCP-Verbindungen aufbauen können.
- Napster identifiziert Clients über Nicknamen, nicht Socket-Adressen[8]. Das bedeutet, dass zwischen Suche und Download der Client seine Adresse ändern kann.
- Napster unterstützt eine rudimentäre Chat-Funktion.

7 in Bezug auf das „Masquerading“, einer Technologie, um Rechnern mit privaten IP-Adressen Zugang zum öffentlichen Internet zu ermöglichen, indem eine fremde, öffentliche IP-Adresse als Absender benutzt wird.

8 Eine Socket-Adresse ist im Internet die Kombination aus IP-Adresse und Port (sowie implizit dem gemeinten Protokoll, also zum Beispiel TCP oder UDP).

- Napster unterstützt eine rudimentäre Instant-Messenging-Funktion in der Hinsicht, dass ein Client mit Nickname A ausdrücklich informiert werden kann, wenn sich der Client mit Nickname B beim Server anmeldet.
- Clients melden dem Server, wann ein Download bzw. Upload anfängt und aufhört. So kann sich der Server die Zahl der Uploads und Downloads merken, welche zur Bewertung von Nutzern herangezogen werden kann.
- Es können Teile von Dateien heruntergeladen werden.
- Napster war anfangs nur auf MP3-Dateien ausgerichtet. Später kam eine rudimentäre Unterstützung für WMA-Dateien hinzu. Weitere Dateitypen werden nicht unterstützt.
  - Für jede Musik-Datei wird ein Hash-Wert, die Größe, Bitrate[9], Sampling-Frequenz und Länge angegeben.
  - Anhand zwei Substrings im Dateinamen, der Maximal-Bandbreite des Lieferanten, der Bitrate und Sampling-Frequenz kann man Dateien suchen.
  - Der Hash-Wert wird nur für die ersten 300KB einer jeden Datei berechnet. Für WMA-Dateien wird er gar nicht berechnet. Damit können viele unterschiedliche Dateien gleiche Hash-Werte haben.
- Das Napster-Protokoll ist nicht eindeutig und exploitbar.
  - Beim Download zwischen Clients ist beispielsweise nicht genau klar, welche Bytes noch zur Angabe der Datei-Länge gehören und welche Bytes schon zum Anfang der eigentlichen Datei gehören.
- Dateien werden anhand ihres Dateinamens identifiziert. Das heißt, dass sie nicht anhand ihres Inhalts identifiziert werden. Das heißt,
  - dass Napster nicht mit zwei Dateien gleichen Namens auf dem gleichen Client umgehen kann.
  - dass die durch Popularität einer Datei entstehende Replikation nicht automatisch ausgenutzt wird.
- Clients können ihre maximale Bandbreite angeben, jedoch nicht beliebig, sondern nur in Stufen 0..10, welche nicht linear mit der realen maximalen Bandbreite korrelieren.
- Das Napster-Protokoll ist nicht offen gelegt. Das Client-Server- und Client-Client-Protokoll wurde teilweise durch Rückentwicklung bekannt. Das Server-Server-Protokoll ist unbekannt.
- Das Napster-Protokoll ist nicht verschlüsselt.

Schlussfolgerung

Man kann sagen, dass Napster sein Ziel, MP3-Dateien von Usern untereinander auszutauschen, mehr oder weniger erreicht hatte. Da das Netz doch sehr leicht zerstörbar ist und in der Geschichte auch zerstört wurde, ist es offensichtlich zu Fehler-intolerant. Da das Protokoll nicht offen gelegt ist (und es im Protokoll auch nicht vorgesehen ist), können Nutzer nicht zur Skalierbarkeit beitragen. Wegen der Design-Fehler, Suboptimalilät und Eingeschränktheit auf Musik-Dateien ist die Napster-Architektur nicht zur Weiterentwicklung geeignet.

9 also die benötigte Bandbreite, wenn man das Stück in Echtzeit hören möchte

4. Gnutella

Am 14. März 2000 wurde die erste Gnutella-Software von Justin Frankel und Tom Pepper veröffentlicht, um zu zeigen, dass Filesharing ohne eine zentrale Instanz auskommen kann. Schon am folgenden Tag wurde die Gnutella-Software auf Weisung des Arbeitgebers von Frankel (AOL) vom Netz genommen. Dies tat der Popularität von Gnutella jedoch keinen Abbruch. Nach ein paar Tagen war das Gnutella-Protokoll rückentwickelt und es fingen OpenSource-Clients an aufzutauchen.

Architektur

Im Gnutella-Netzwerk können alle Teilnehmer alle Rollen gleichzeitig übernehmen:

- Datei senden
- Datei empfangen
- Router
- Such-Anfrage Aussenden
- Such-Anfrage Beantworten
- Netzwerk-Teilnehmer finden

Es ist recht erstaunlich, dass dazu nach dem usprünglichen Gnutella-Protokoll nur 4 Pakettypen ausreichen:

- *Ping*: Wunsch nach Information über Teilnehmer
- *Pong*: Information über Teilnehmer (enthält Socket-Adresse)
- *Query*: Suche nach Dateien mit angegebenen Schlüsselwörtern im Namen
- *QueryHits*: Liste von Such-Ergebnissen
  - enthält Socket-Adresse des Lieferanten.
  - enthält Teilnehmer-ID des Lieferanten.
  - enthält eventuell optionale Daten.
  - Jedes Such-Ergebnis besteht aus
    - einer Datei-Nummer bezüglich des Lieferanten,
    - der Datei-Größe,
    - dem Dateinamen,
    - und eventuell optionalen Daten.

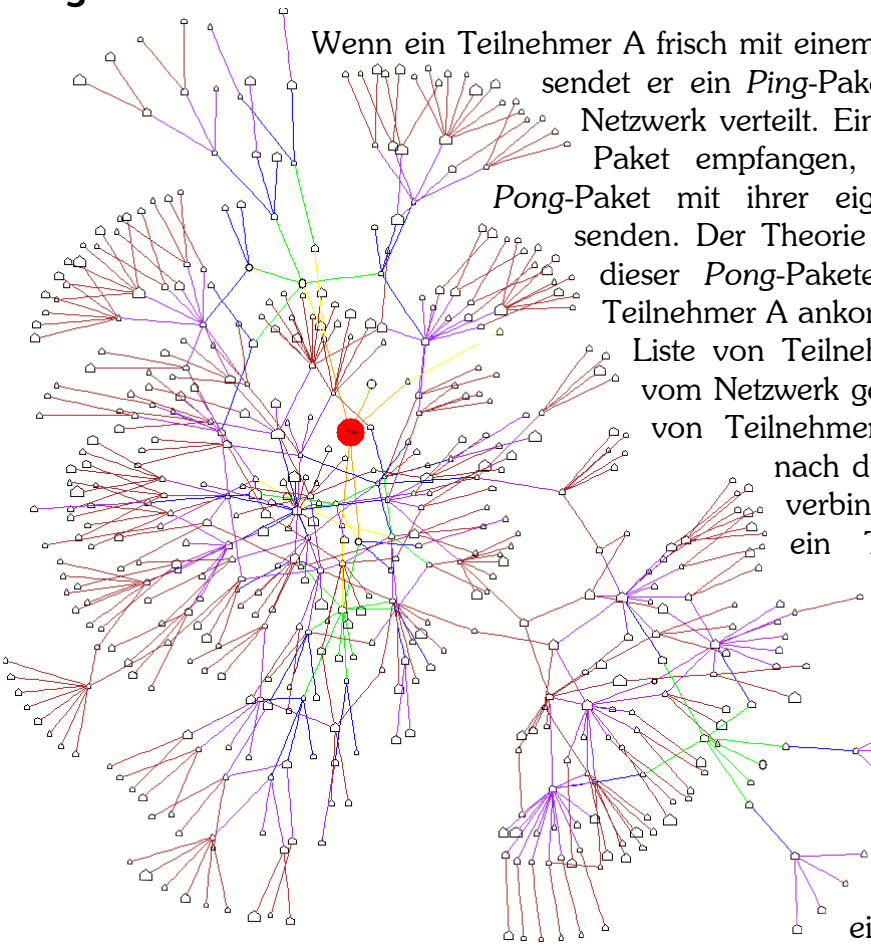
Alle Pakete haben eine pseudo-eindeutige Identifikationsnummer sowie einen

- Time-to-live-Zähler, der angibt, wieviele Male das Paket noch weitergeleitet werden darf, und einen
- Hop-Zähler, der angibt, wieviele Male das Paket schon weitergeleitet wurde.

Ähnlich wie beim Internet-Protokoll wird bei jedem Empfang der Time-to-live-Zähler um 1 dekrementiert. Der Hop-Zähler wird bei jedem Empfang um 1 inkrementiert.

Der allgemeine Routing-Algorithmus im Gnutella-Netzwerk ist (keiner): Fluten („flooding“). Das heißt, dass ein empfangenes Paket an alle bestehenden Verbindungen weitergeleitet wird (außer zu der Verbindung, über die es direkt empfangen wurde). Eine kleine Standard-Time-to-live von 7 und das kurzfristige Speichern der Identifikationsnummer sollen herumkreisenden Paketen vorbeugen.

Vorgehensweise



Wenn ein Teilnehmer A frisch mit einem anderen verbunden ist, dann sendet er ein *Ping*-Paket aus. Dieses wird über das Netzwerk verteilt. Einige Teilnehmer, die ein *Ping*-Paket empfangen, entscheiden sich dazu, ein *Pong*-Paket mit ihrer eigenen Socket-Adresse auszusenden. Der Theorie nach müsste der größte Teil dieser *Pong*-Pakete auch beim ursprünglichen Teilnehmer A ankommen.[10] Folglich erhält er eine Liste von Teilnehmern. Ist Teilnehmer A nun vom Netzwerk getrennt, so kann er seine Liste von Teilnehmern („*host cache*“) der Reihe nach durchgehen, um sich wieder zu verbinden. Typischerweise versucht ein Teilnehmer immer, 4 Verbindungen gleichzeitig aufrecht zu erhalten.

Möchte Teilnehmer A nun nach einer Datei suchen, so sendet er ein *Query*-Paket mit einer entsprechenden Liste von Schlüsselwörtern aus. Meint ein potentieller Lieferant auf ein solches Paket antworten zu wollen (weil er beispielsweise noch Upload-Kapazität frei hat), dann sendet er eine Reihe von *QueryHits*-Paketen aus. Sind diese bei Teilnehmer A angekommen, kann nun die Ergebnis-Listen aller empfangenen *QueryHits*-Pakete vereinigen und diese Liste anhand der eigenen Schlüsselwörter filtern.

Hat sich Teilnehmer A für ein Ergebnis entschieden, so startet er den eigentlichen Download, indem er sich auf die Socket-Adresse des Lieferanten verbindet und die Datei per HTTP anhand der Dateinummer anfordert.

Finden von Teilnehmern

Während bei Napster klar war, wie der Client zum Netzwerk findet (nämlich durch einen vorgegeben, bekannten Server), ist dies bei Gnutella nicht mehr so klar. Selbst die Entwickler von Teilnehmer-Software müssen keinen bekannten Teilnehmer laufen lassen.

Bekannte Discovery-Teilnehmer

Trotzdem war dies natürlich der Fall: Entwickler und einige unabhängige Interessierte ließen spezielle Teilnehmer-Software laufen, die weder Dateien bezieht noch freigibt. Statt dessen war

10 Meist ist das eigene Aussenden von *Ping*-Paketen nicht erforderlich, da meist schon ein anderer Teilnehmer ein *Ping*-Paket ausgesendet hat und man die dazugehörigen *Pong*-Pakete bereits mit empfängt.

deren Aufgabe einzig und allein, sich mit möglichst vielen Teilnehmern zu verbinden und so alle möglichen gültigen Socket-Adressen von Teilnehmern in Erfahrung zu bringen.
Hatte sich nun ein normaler Teilnehmer A mit diesem Discovery-Teilnehmer B verbunden, so sendete B nur *Pong*-Pakete an A mit entsprechenden gültigen Socket-Adressen. Anschließend wurde die Verbindung sofort wieder getrennt.
Der Sinn einer solchen Aktion ist es, den *host cache* von A mit aktuellen Socket-Adressen zu füllen. Dazu muss die Adresse des Discovery-Teilnehmers B aber bekannt sein. Solche Teilnehmer waren früher ausreichend häufig, sind heutzutage jedoch kaum noch zu finden.

Gespeicherter Host Cache

War ein Teilnehmer verbunden mit dem Netzwerk, so kann er seinen *host cache* auf der Festplatte speichern. Beim nächsten Start stehen die Chancen nicht schlecht, dass einige wenige der Einträge dieses *host cache*s noch gültig sind.

GWebCache

In Mode gekommen ist ein HTTP-Server-Skript namens GWebCache[11]. Verschiedene Instanzen dieses Skripts sind über viele Web-Server verteilt. Teilnehmer mit relativ stabiler Socket-Adresse können das Skript aufrufen, um auszusagen, dass sie

- selbst existieren und erreichbar sind und
- weitere funktionierende Adressen von GWebCaches kennen.

Diese Informationen werden dann vom Skript gespeichert. Teilnehmer, die einen leeren *host cache* haben, können aus einer Liste bekannter GWebCache-URLs eine Adresse auswählen und diese abfragen. Das Skript liefert eine Liste von Socket-Adressen sowie eine Liste von GWebCache-URLs. Da URLs wesentlich langlebiger sind als Socket-Adressen sinkt die Wahrscheinlichkeit drastisch, dass ein Teilnehmer das Netzwerk nicht findet.

GWebCache Data

[Home | Statistics | Data]

Hosts	Alternate Caches
142.173.37.142:6349	http://www.theholt.net/cache.php
64.228.78.82:6346	http://gwebcache.zerouptime.ch/gcache.php
216.228.181.4:6346	http://ims.ecn.purdue.edu/~mckeowbc/gcache.php
209.142.138.128:6346	http://usuarios.lycos.es/coolebra/gwebcache/gcache.php
213.103.95.140:6346	http://www.wtndrifters.com/gcache.php
24.46.229.128:11138	http://mitglied.lycos.de/phpgk
65.27.155.94:6348	http://www.apexo.de/gnucache/gcache.ph
64.83.64.70:6346	http://corky.net/gcache/gcache.php
80.186.117.28:8436	http://209.197.225.202/gcache.php
212.195.244.61:6346	http://members.lycos.co.uk/petera2003/cache_thingy/gcache.php
82.66.13.168:6346	
205.184.203.25:23364	
68.146.189.171:6346	
24.193.6.212:6346	
67.162.196.133:6346	
66.72.173.8:6348	
208.154.237.151:6346	
80.235.142.57:6346	
81.53.237.79:6346	
66.189.11.142:7407	

Auszug aus http://gwebcache.bearshare.net/gcache.php?data=1

Bemerkungen

Das originale Gnutella-Protokoll hat wegen seiner Einfachheit relativ viel Aufmerksamkeit auf sich gezogen, aber auch gehörige Design-Probleme:

- So sind alle Teilnehmer hinter dem Horizont von (in der Regel) 7 Hops effektiv nicht erreichbar.
- Die Bandbreiten-Effizienz leidet deutlich unter der Paket-Überschwemmung.
- Das Netz riskiert mit der Anzahl der Teilnehmer quadratisch wachsende Last, da es effektiv wie ein Broadcast-Medium oder eine Menge von Punkt-zu-Punkt-Verbindungen zwischen

11 Erhältlich unter http://www.gnucleus.com/gwebcache/

jedem Teilnehmer wirkt, wenn es nicht die Netz-Zerteilung durch den Horizont eines jeden Teilnehmers gäbe.

Anderenseits ergeben sich durch dieses Design auch interessante Eigenschaften:
- So lässt sich eine passive Suche bewerkstelligen, indem einfach nur den durchrauschenden *QueryHits*-Paketen gelauscht wird.
- Die Ausfall-Sicherheit ist vergleichsweise sehr gut. Es müssten schon alle Socket-Adressen ausfallen, die ein Teilnehmer jemals empfangen hat (über *Pong*- oder *QueryHits*-Pakete), damit der Teilnehmer nicht mehr weiß, mit welchem Teilnehmer er sich verbinden sollte. Typischerweise ist dieser *Host Cache* über 20'000 Einträge lang.

Allgemein hat das Gnutella-Protokoll unter Wildwuchs zu leiden. Viele Clients versuchen ihre halb-privaten Erweiterungen mit einzubauen. Jeder Paket-Typ hat Platz für Erweiterungen. Diese werden jedoch nicht in einheitlicher Weise benutzt.
- So ist nicht klar, wie die Schlüsselwörter in einem Query-Paket zu interpretieren sind. Weder der Zeichensatz (z.B. UTF-8, ISO-8859-15, ...) noch die Relation zwischen Schlüsselwörtern und freigegebenen Dateien (z.B. „Und"-Verküpfung, „Oder"-Verknüpfung, genau diese Wort-Reihenfolge, ...) sind eindeutig definiert.
- So sendet *LimeWire*[12] XML-Metadaten in den *QueryHits*-Paketen, deren Format nicht standardisiert ist.
- So sendet *BearShare*[13] auch Metadaten in *QueryHits*-Paketen, jedoch in einem anderen Format.
- Es sind andere Verwendungen von optionalen Datenbereichen bekannt, so zur Angabe eines URNs (UniformResourceName) der beschriebenen Datei.
- Es gibt eine Reihe von weiteren verwendeten Paket-Typen. Ein sehr weit verbreiteter Paket-Typ ist das *Push*-Paket:
  - Kann ein potentieller Lieferant keine eingehenden Verbindungen annehmen, so kann er mittels eines *Push*-Pakets aufgefordert werden, eine ausgehende Verbindung zum Bezieher aufzubauen.

Inzwischen wird versucht, den Protokoll-Wildwuchs zu standardisieren und das Protokoll zu erweitern. Eine Erweiterung sind die *Ultrapeer*s, also Teilnehmer mit zentraleren Aufgaben. Damit sollen die Skalierbarkeitsprobleme reduziert werden.

Schlussfolgerung

Gnutella hat das ursprüngliche Ziel - zu zeigen, dass Filesharing ohne zentrale Instanz auskommen kann - erreicht. Dem heutigen Stand der Wissenschaft entspricht das ursprüngliche Protokoll jedoch nicht. Vor allem die quadratisch anwachsenden Kosten vereiteln das Wachstum des Netzes zum Nutzen für alle Teilnehmer. DateiIDs (von der Datei-Größe einmal abgesehen) sind unbekannt, Nutzer-Authentifizierung und Teilnahme-Belohnung sind nicht vorgesehen, Suchen sind unvollständig. In vieler Hinsicht ist Gnutella sogar nachteiliger als Napster. Aber Napster ist längst abgeschaltet, und Gnutella lebt immer noch.

12 Eine Gnutella-Teilnehmer-Software
13 Eine Gnutella-Teilnehmer-Software

5. FastTrack-Protokoll

Das FastTrack-Protokoll kommt in der Filesharing-Software „KaZaA“ und „Grokster“ zum Einsatz. Es ist proprietär und verschlüsselt. Deswegen entzieht es sich der allgemeinen Analyse. Als einst das Protokoll fast analysiert war, wurde es kurzerhand geändert und die Änderung über eine automatische Update-Funktion in den Filesharing-Anwendungen forciert. Deswegen ist recht wenig über dieses Protokoll bekannt. Die Behandlung des FastTrack-Protokolls muss daher leider ausfallen.

Bekannt ist jedoch, dass einige Teilnehmer die Rolle „*supernode*“ übernehmen, die als Verzeichnis für die anderen Teilnehmer agieren. Demnach scheint das Protokoll ähnlich dem eDonkey-Protokoll zu sein.

Ebenfalls ist bekannt, dass Teilnehmer zwar einen Hash-Wert über eine Datei berechnen, dies jedoch wieder nur für einen Anfang eine Datei. Aus diesem Grund soll die „Ausschuss-Rate“ recht hoch sein. Das heißt, dass oft Dateien verschiedenen Inhalts zusammengemischt werden.

6. eMule

Es gab einmal eine Filesharing-Software namens „eDonkey 2000“. Deren Protokoll ist ebenfalls proprietär, wurde jedoch inzwischen weitgehend rückentwickelt. Für dieses Protokoll wurden OpenSource-Implementationen geschrieben. Prominentester Vertreter ist die Software „eMule“[14]. Diese hat dem Protokoll einige Erweiterungen hinzugefügt, weswegen im Folgenden auch vom „eMule-Protokoll“ die Rede ist. Leider ist das Protokoll trotzdem nicht sauber dokumentiert. Es lässt sich jedoch folgendes sagen:

Das eMule-Protokoll ist asymmetrisch. Das heißt, dass ein Teilnehmer im Netzwerk nicht automatisch alle Rollen übernehmen kann. Vielmehr gibt es Clients und mehrere Server. Es ist derzeit keine OpenSource-Implementation für die Server-Rolle bekannt.

Das ursprüngliche Protokoll war durchaus Napster-ähnlich. Das heißt, dass der Client
- sich zu einem Server verbindet,
- ihm die eigenen freigegebenen Dateien meldet,
- ihm Such-Anfragen schickt und
- Lieferanten ausfindig macht.

Es ist nicht bekannt, dass Server untereinander mehr kommunizieren, als sich gegenseitig von ihrere Existenz zu informieren. Insbesondere werden Such-Anfragen offensichtlich nicht zwischen den Servern weitergeleitet. Somit haben Anfragen auch hier einen Horizont, und zwar den des Servers, mit dem der Client gerade verbunden ist.

eMule bietet gegenüber Napster einige zusätzliche Features:

14 erhältlich unter http://www.emule-project.net/

Server-Liste

Hat sich ein Client mit einem Server verbunden, so erhält der Client eine aktuelle Liste von Servern. Damit ist eine gewisse Ausfallsicherheit gegeben, muss sich ein Client doch nur mehr oder weniger regelmäßig mit einem der Server seine Liste verbinden, um die Liste aktuell zu halten.

DateiIDs

Von jeder Datei wird ein MD4-Hash-Wert erzeugt, welcher als pseudo-eindeutige Identifikationsnummer dient. Nachdem der Client eine Such-Anfrage an den Server geschicklt hat, antwortet der Server mit einer Liste von Metadaten über passenden Dateien. Diese Metadaten enthalten auch die DateiID der beschriebenen Datei.

Möchte ein Bezieher nun eine Datei herunterladen, so fragt er anhand der DateiID den Server nach einer Liste von Lieferanten für dieser Datei. Der Server antwortet mit einer Liste von Lieferanten.

Mit Hilfe der DateiIDs kann der Vorgang der Suche und der Vorgang des Downloads zeitlich getrennt werden. Es ist also für den Nutzer möglich, den Download von Dateien aus der Such-ergebnis-Liste erst einige Wochen später zu vollziehen.

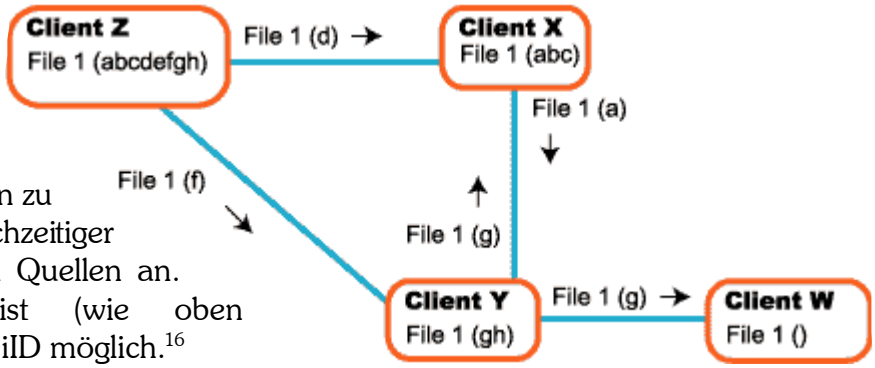
- Dies ist für größere Dateien auch dringend notwendig, denn mit der Größe der Datei steigt die Wahrscheinlichkeit eines Ausfalls des ursprünglichen Lieferanten. Die Erreichbarkeit von Dateien ist längst nicht so volatil wie die Erreichbarkeit von Lieferanten. DateiIDs machen so Downloads von größeren Dateien überhaupt erst erforderlich.
- Zudem erlaubt die Trennung zwischen Suche und Download ein anderes Nutzungs-Verhalten. Nutzer können Dateien suchen und diese zu einer „Wunschliste“ hinzufügen. Je nach Verfügbarkeit dauert es Minuten bis Wochen, bis die gewünschte Datei heruntergeladen wurde. Ist eine Datei selten repliziert, also die Wahrscheinlichkeit klein, dass ein Lieferant gerade diese Datei liefern kann, dann ermöglichen DateiIDs erst den Download seltener Dateien. Denn ohne DateiIDs würden alle erreichbaren Lieferanten mit der Zeit verschwinden.[15]

Segmente

Größere Dateien werden logisch in Segmente zu je 9500KB zerlegt. Über jedes Segment wird separat ein Hash-Wert ermittelt. Dies erlaubt eine feinere Sicherung der Datenübertragung. Diese ist für größere Dateien auch notwendig. Denn sonst kann es sein, dass einer der vielen Lieferanten eine falsche Teil-Datei geliefert hat, aber nicht klar ist, welcher Teil der großen Datei korrupt ist. Die Überprüfung der einzelnen Segmente erlaubt die Eingrenzung der fehlerhaften Stelle in der Datei. Ist das korrupte Segment identifiziert, dann muss nur noch dieses neu heruntergeladen werden, anstatt die große Datei komplett neu herunterzuladen.

15 Vom Wiederauftauchen von Lieferanten werden Bezieher in der Regel nicht informiert. Der Bezieher könnte die ursprüngliche Suchanfrage später noch einmal absenden, um eine neue Lieferanten-Liste zu erhalten, jedoch ist nicht automatisch klar, welche Datei der neuen Ergebnisliste zu welcher Datei der alten Ergebnisliste gehört. Demnach wäre bei der Wiederaufnahme eines Downloads nicht gesichert, dass auch wirklich der Download der die selbe Datei fortgesetzt werden würde.

Paralleler, kooperativer Download



Oft ist es so, dass die ausgehende Bandbreite eines Lieferanten kleiner ist als die eingehende Bandbreite des Beziehers. Um diese Ausnutzen zu können, bietet sich ein gleichzeitiger Download aus verschiedenen Quellen an. Diese zu identifizieren ist (wie oben beschrieben) anhand der DateiID möglich.[16]
Interessant ist der implizit kooperative Download. Statt linear vom Anfang der Datei bis zu ihrem Ende herunterzuladen, wählen sich die meisten eMule-Clients ein zufälliges Fragment der gewünschten Datei beim Lieferanten aus. Damit ist die Wahrscheinlichkeit, dass ein bestimmtes Fragment im Netzwerk vorhanden ist, ausgeglichener. Hat ein Teilnehmer X bereits ein Fragment fertig heruntergeladen, so kann er es einem Bezieher Y der selben Datei zur Verfügung stellen. Dieser wiederum kann dem Teilnehmer X die Fragmente liefern, die er bereits hat.
Der kooperative Download erhöht also sowohl die Verfügbarkeit einer kompletten Datei als auch ihre Download-Geschwindigkeit.

Lieferungs-Warteschlange

Da die ausgehende Bandbreite begrenzt ist, ist klar, dass nicht beliebig viele Lieferungen gleichzeitig sinnvoll sind. Vielmehr gibt es eine Anzahl von gleichzeitigen Lieferungen, die ungefähr das Optimum darstellen (zum Beispiel 4). Die Frage ist nun, was passiert, wenn mehr Lieferungs-Anfragen kommen als Lieferungen derzeit ausgeführt werden sollen. Napster- und Gnutella-Teilnehmer lehnen diese Anfragen einfach ab. Stattdessen führen eMule-Clients Lieferungs-Warteschlange. Dies ist eine Liste von Teilnehmern, die von diesem Client ein Datei-Fragment beziehen wollen.
Damit wird die Netzwerk-Last reduziert, die entstehen würde, wenn der Lieferant immer vom selben Bezieher gefragt werden würde, ob nun eine Datei endlich lieferbar wäre.

Gegenseitige Credits

Normalerweise werden die Bezieher in der Warteschlange in der Reihenfolge bedient, wie sie ankommen („first come, first serve“). Jedoch haben neuere eMule-Clients ein Credit-System: Lädt Teilnehmer A von Teilnehmer B herunter, so merkt sich A die Menge der empfangenen (und auch gesendeten) Daten. Da A von B profitiert hat, hat A gegenüber B eine Schuld. Möchte nun B von A ein Fragment empfangen, so erinnert sich A an diese Schuld mit der Folge, dass B in der Warteschlange schneller nach vorne schreiten darf. Da die Schuld beim Schuldner gespeichert wird, kann sie nur dort manipuliert werden. Daran hat der Schuldner jedoch kein Interesse, denn wer an welcher Position in der Warteschlange ist, kann ihm ziemlich egal sein. Ist jedoch ein Gläubiger in der Warteschlange, dann erhöht die Geschwindigkeit, ihn zu bedienen, die Wahrscheinlichkeit, selbst von ihm später bevorzugt bedient zu werden.

16 Dies ist an sich nichts besonderes. Fortgeschrittene Gnutella-Implementationen können dies auch, auch wenn sie sich mit Datei-Größe und Datei-Namen als Kriterium für Datei-Gleichheit begnügen müssen. Oft haben die selben Dateien jedoch unterschiedliche Namen, sodass Gnutella hierbei nicht so effizient ist wie eMule.

- Damit wird die Teilnahme am parallelen, kooperativen Download belohnt. Wer selbst kein Fragment einer gemeinsam gewünschten Datei anbietet, kann schwerer am gemeinsamen Download teilnehmen, als jemand, der wirklich kooperativ ist.
- Der Kooperations-Effekt ist aber nicht auf den kooperativen Download einer einzigen Datei beschränkt. Haben zwei Nutzer die gleichen Interessen, so steigt die Wahrscheinlichkeit, dass sie sich beim Download einer Datei gegenseitig helfen können, was sie dann auch dank der Credits beschleunigt tun.

eMule verwendet digitale Signaturen (mit privaten und öffentlichen Schlüsseln) für die eindeutige Identifizierung von Gläubigern. Somit wird ausgeschlossen, dass jemand versucht, die Credits eines anderen auszunutzen.

Horizont-Wanderung

Wie eingangs erwähnt hat jeder Client einen Horizont, der durch den Server und dessen verbundene Clients begrenzt ist. Clients, die mit einem anderen Server verbunden sind, sind so erst einmal nicht erreichbar. Jedoch geht die Verbindung zwischen Client und Server nach einiger Zeit verloren, sodass dieser Client sich neu verbinden möchte. Die Wahrscheinlichkeit ist nicht gering, dass er sich anhand seiner Server-Liste mit einem anderen Server als vorher verbindet. Damit wandern jedoch alle freigegebenen Dateien und Fragmente ebenfalls von einem Horizont in einen anderen. Auf diese Weise wird für Diffusion von Fragmenten zwischen den verschiedenen Teil-Netzen gesorgt.

Lieferanten-Listen-Austausch

Neuere eMule-Clients A sind in der Lage, nach der Kontaktierung eines Lieferanten B die Liste aller ihm bekannten Lieferanten D_0, D_1, ... abzufragen. Auf dem ersten Blick erscheint dies wenig sinnvoll, muss der Lieferant B doch zum selben Teil-Netz gehören, zu dem der Bezieher A auch gehört, damit ihn der Bezieher A hat finden können. Gehört A zum selben Teil-Netz, so kennt er bereits alle Lieferanten C_0, C_1, ... , da diese bereits der Server angezeigt wurden.
Jedoch kann es sein, dass der Lieferant B eine Horizont-Wanderung gemacht hatte, als er schon selbst eine Lieferanten-Liste D_0, D_1, ... für diese Datei hatte. Diese Lieferanten-Liste D_0, D_1, ... kann also andere Lieferanten enthalten als die bereits bekannte Liste C_0, C_1, ... , da die neuen Lieferanten aus einem anderen Teilnetz stammen. Diese Lieferanten kann man nun wieder nach weiteren Lieferanten fragen. Auf diese Weise lässt sich für die Zwecke des Downloads (nicht aber für die Zwecke des Suchens) der Horizont überspringen.

Dateinamen-Austausch

Häufiger werden Dateien umbenannt. Wie schon gezeigt, kann dies dem eMule-Netzwerk wegen der DateiIDs nichts anhaben. Möglicherweise ist Umbenennen hier sogar von Vorteil: Die Wahrscheinlichkeit wird größer, eine Datei zu finden, wenn sie verschiedene gleichbedeutende Namen hat, der Benutzer aber nur eines dieser Synonyme eingegeben hat.

Beispiel

Die Beethovens 5. Sinfonie lässt sich sowohl unter „symphony“ als auch unter „sinfonie“ finden, wenn die selbe Datei von verschiedenen Nutzern je nach ihrer Sprache unterschiedlich benannt wurde.

Auf eine Download-Anfrage mittels DateiID hin liefern potentielle Lieferanten einen ihnen bekannten Dateinamen für diese Datei. Da sich ein Bezieher bei möglichst vielen Lieferanten in der Warteschlange einträgt, erhält er für die selbe Datei eine Liste von Dateinamen für diese Datei. Nutzer können nun anhand der Dateinamen schließen, ob die Datei ein *Fake* ist, also eine Datei, von der man annimmt, dass sie einen bestimmten Inhalt hat, dies aber gar nicht der Fall ist. Denn ist eine Datei ein *Fake*, hat sie oft sehr unterschiedliche Dateinamen, welche keine Synonyme mehr sind.

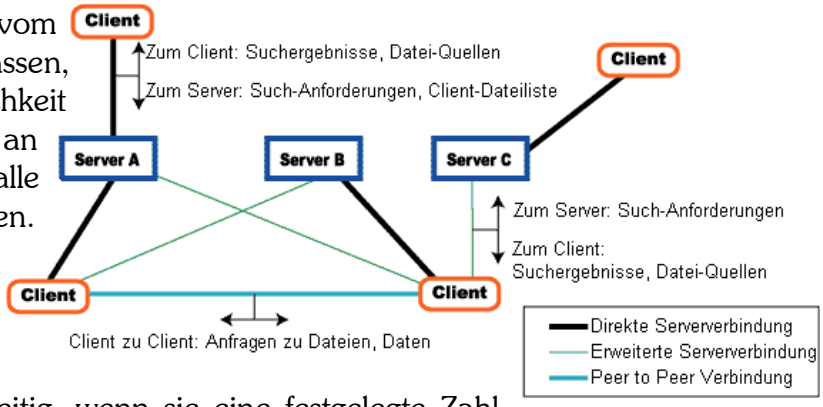
Bemerkungen

- Anders als bei Napster, KaZaA oder Gnutella wird im eMule-Netzwerk über die gesamte Datei-Länge ein Hash-Wert gebildet. Jedoch ist der verwendete Hash-Algorithmus MD4. Dieser wird inzwischen als kryptographisch unsicher angesehen. Das heißt, dass es nur Minuten dauert[17], um eine Datei zu erzeugen, die den selben MD4-Wert hat, aber einen anderen Inhalt. Möchte jemand also die Verbreitung einer bestimmten Datei verhindern, so hat er hier einen passenden Ansatzpunkt, indem er sich selbst am Netzwerk beteiligt und so viele falschen Kopien verbreitet, wie nur möglich. Die korrekten Kopien gehen dann im Meer der Fälschungen tendenziell unter.
- eMule-Clients setzen oft Beschränkungen der Empfangs-Bandbreite abhängig von der maximalen Sende-Bandbreite. Je kleiner die maximale Sende-Bandbreite vom Nutzer eingestellt wird, desto kleiner wird die maximale Empfangs-Bandbreite, die vom Client akzeptiert wird. Dies soll „antisoziales Verhalten“ (dass also nur vom Netzwerk bezogen aber nicht geliefert wird) verhindern. Da die meisten Nutzer eine erhebliche Schwelle haben, den Quellcode zu ändern, wirkt diese Beschränkung erstaunlich gut. Clients, von denen bekannt ist, dass sie keine solche Beschränkung unterstützen, werden teilweise überhaupt nicht in die Warteschlange aufgenommen.
- Durch die Warteschlange werden Teilnehmer, die länger warten, den Teilnehmern, die kürzer warten, bevorzugt. Dies wiederum bedeutet, dass es teilweise Stunden bis Tage dauert, bis ein Bezieher eine Lieferung erhält. Dies wiederum bedeutet, dass nur Nutzer mit ständiger Netzwerk-Verbindung und hoher Datenvolumen vom Netzwerk profitieren können. Deswegen ist hier, anders als bei Gnutella, eine Pauschaltarif für die Internet-Nutzung effektive Zugangsvoraussetzung.
- Clients werden bei eMule nur mittels der IP-Adresse (nicht etwa mit einer Socket-Adresse) identifiziert. Das heißt, dass pro IP-Adresse nur eine Client-Instanz laufen kann, ohne dass es zu verwechslungen kommt.
- Auch eMule unterstützt „maskierte“ Teilnehmer, die keine eingehenden Verbindungen haben können.[18] Diese allerdings werden sowohl von Servern als auch anderen Clients benachteiligt, indem die Server die Anzahl von maskierten Teilnehmern begrenzen und die anderen Clients die maskierten Teilnehmer langsamer in ihrer Warteliste fortschreiten lassen. Grund ist, dass die maskierten Teilnehmern den Servern überproportional viel Arbeit machen, da die ganze Koordination für einen Download indirekt über den Server laufen muss. Deswegen werden sie ebenfalls als „antisozial“ eingestuft.

17 jedenfalls http://www.rsasecurity.com/rsalabs/faq/3-6-6.html

18 Diese werden als „Low ID“-Teilnehmer bezeichnet. Dies rührt daher, dass die IP-Adresse des Clients als temporäre 32-Bit-Identifikationsnummer verwendet wird. Hat ein maskierter Teilnehmer keine echte IP-Adresse, dann vergibt der Server einfach eine Nummer zwischen 0x00000000 und 0x00FFFFFF.

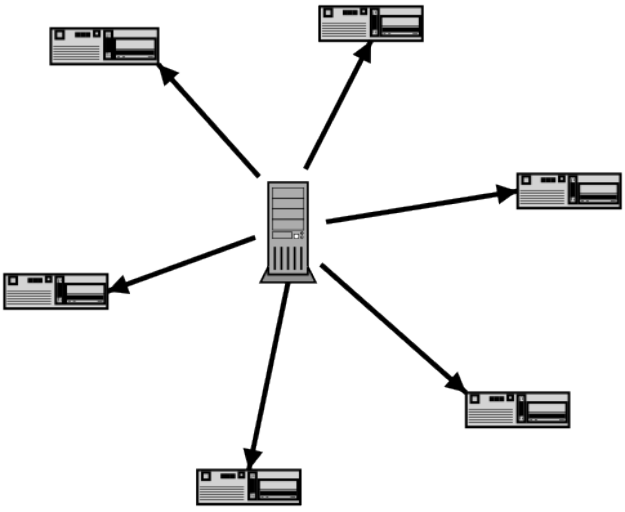
- Da Suchanfragen nicht das vom Server gebildete Teilnetz verlassen, bieten einige Clients die Möglichkeit an, eine Suchanfrage statt nur an den aktuellen Server an alle bekannten Server zu schicken. Dies wird ebenfalls teilweise als „antisozial“ angesehen, da die Server-Last dadurch steigt. Die meisten Clients beenden jedoch die Anfrage-Welle vorzeitig, wenn sie eine festgelegte Zahl von Ergebnissen (z.B. 200) bereits empfangen haben.



Schlussfolgerung

Trotz der Client-Server-Architektur ist das eMule-Netzwerk durchaus brauchbar. Man soll auch nicht verkennen, dass es Server gibt, die über 250'000 User gleichzeitig aufnehmen[19]. Bei dieser Kapazität relativieren sich die Client-Server-Probleme, wie etwa ein zu kleiner Horizont. Andere Filesharing-Netzwerke sind möglicherweise skalierbarer, jedoch haben sie überhaupt nicht mehr als 250'000 Nutzer, welche die Teilnahme an skalierbareren Netzwerken überhaupt attraktiver machen würde, als eMule zu benutzen. eMule hat viele interessante und nützliche Features geliefert, die die *Leech*-Problematik angehen und Filesharing von großen Dateien erst ermöglicht. [20]
Da jedoch die Server-Software keine Freie Software ist (und jeder Server die selbe Software benutzt), ist zu befürchten, dass die einigen dutzend Server eine noch zu konzentrierte Angriffsfläche bieten, als dass man das eMule-Netzwerk als unverwundbar ansehen könnte.

7. BitTorrent



Anders als bei den vorgenannten Filesharing-Netzwerke liegt bei BitTorrent nicht der Focus auf Filesharing, sondern auf Distribution. Und zwar liegt BitTorrent der Gedanke zu Grunde, dass FTP- oder HTTP-Server allein häufig überlastet sind, wenn es gilt, eine größere, populäre Datei (beispielsweise eine neue Linux-Distribution) herunterzuladen. Bei Lichte betrachtet hat nämlich nicht nur der Server die populäre Datei. Auch die Clients, die gerade die Datei herunterladen, haben zumindest Teile dieser Datei. Zudem ist es oft so, dass diese Clients Sende-Bandbreite übrig haben. BitTorrent versucht diese auszunutzen.

19 Bekannte Server und deren Last werden zum Beispiel von http://ed2k.2x4u.de/list.html angezeigt.
20 Obwohl allgemein gesagt wird, dass eMule für große Dateien ausgelegt sei, unterstützt das derzeitige eMukl-Protokoll nur Dateien bis 4GB Größe. Dies wird für die Zukunft bald zu wenig sein.

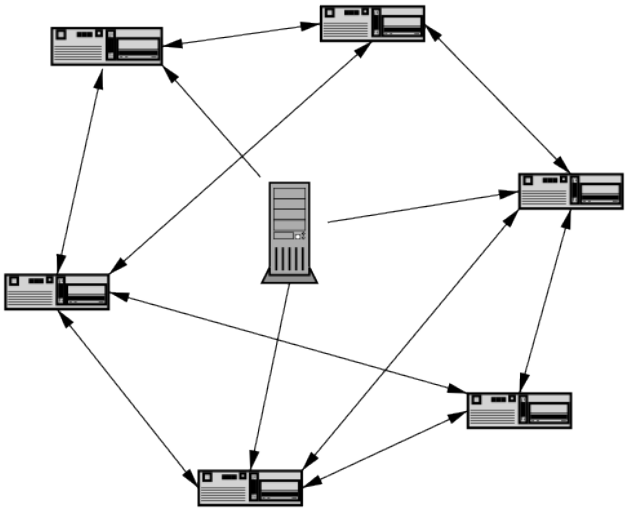
Architektur

Dabei baut BitTorrent für jede zu verbreitende Datei ein eigenes Filesharing-Netzwerk auf, was strukturell einem eMule-Netzwerk nicht unähnlich ist. Jedes BitTorrent-Netzwerk besteht aus einem zentralen *Tracker* sowie mindestens einem Client. Der *Tracker* ist öffentlich bekannt und unter einer URL erreichbar. Er hilft beim Auffinden der Clients:

- Will ein Client eine Datei über BitTorrent herunterladen, so kontaktiert er den Tracker. Dieser sendet dem Client eine Liste von Clients zu, von denen bekannt ist, dass sie Teile der gewünschten Datei haben.
- Der Tracker wird informiert,
  - wenn der Client den Download beginnt,
  - wenn der Client den Download beendet oder abbricht,
  - regelmäßig während des Downloads, dass der Client noch lebt.

Somit hat der Tracker für die gewünschte Datei ständig einen Überblick über alle Teilnehmer im Filesharing-Netzwerk für diese spezifische Datei.

Paralleler, kooperativer Download



Die Clients kontaktieren sich nun untereinander und starten einen kooperativen Download. Dabei wird die zu beziehende Datei in einzelne Segmente (zum Beispiel zu je 256KiB) aufgeteilt. Clients teilen sich gegenseitig mit, welche Segmente sie beziehen wollen (also noch nicht bezogen haben). Hat Client A ein bestimmtes Segment, was Client B beziehen möchte, so kann Client A entscheiden, dieses an Client B zu liefern. Typischerweise ist die Nachfrage höher als das Angebot. Das heißt, dass Client A die Qual der Wahl hat, welchen Client es nun beliefern soll und welchen nicht. Diese Entscheidung wird von einem komplexen Algorithmus ermittelt, der auf der „Tit-for-tat"-Strategie aufbaut, die aus der Spieltheorie bekannt ist. Im Wesentlichen lässt sich dies als „Wie du mir, so ich dir." zusammenfassen. So erwartet also Client A, nachdem er ein Segment an B geliefert hat, dass er anschließend (oder auch schon gleichzeitig) von B ein Segment geliefert bekommt. Ist dies nicht der Fall, wird B bei folgenden Entscheidungen nicht mehr bevorzugt. Auf diese Weise erreichen kooperative Teilnehmer im Netzwerk hohe Download-Raten, die ähnlich hoch sind wie ihre Upload-Rate. Unkooperative Teilnehmer dagegen müssen tendenziell warten.

Damit die Verbreitung überhaupt starten kann, ist ein Teilnehmer erforderlich, der die Datei vollständig hat. Dieser wird *Origin* genannt.

Web-Integration

Anders als bei den meisten Filesharing-Systemen ist BitTorrent stärker in das World Wide Web integriert. Nutzer, die am kooperativen Download mit teilnehmen möchten, klicken auf einen Link, hinter dem sich eine *.torrent*-Datei verbirgt. Diese enthält die URL des Trackers sowie SHA-1-Hash-Werte über alle Segmente. Der Browser veranlasst dann das Starten einer *helper*

application, also eines Programms, was mit der *.torrent*-Datei etwas anfangen kann. Dieses ist nun der eigentliche Filesharing-Client.

Bemerkungen

- Es fällt auf, dass BitTorrent für das eher private Filesharing nicht geeignet ist. Dies liegt daran, dass
  - die Freigabe relativ kompliziert ist. Es muss sichergestellt werden, dass ständig ein *Tracker* und zumindest anfangs auch ein *Origin* erreichbar ist.
  - es kaum einen Anreiz gibt, Dateien freizugeben, wenn man sie nicht selbst veröffentlichen möchte.
  - nicht geklärt ist, wie Nutzer schnell passende *.torrent*-Dateien finden. Es existieren zwar inzwischen spezialisierte Suchmaschinen dafür, jedoch ist deren Aktualität viel schwächer als bei anderen Filesharing-Systemen.
- Es kommt häufig vor, dass freigegebene Dateien gar nicht mehr verfügbar sind, weil ihre Nachfrage zu weit gesunken und der *Origin* nicht mehr erreichbar ist.
- Das *Tracker*-Konzept erlaubt eine hohe Skalierung. Mit der Nachfrage steigt in der Regel die Verfügbarkeit eine Datei, anstatt sie sinkt wie bei herkömmlichen HTTP- oder FTP-Servern. Jedoch ist die Skalierbarkeit durch den zentralen *Tracker* begrenzt.
- Gleiches gilt für die Angreifbarkeit: der zentrale *Tracker* ist Schwachpunkt des Systems.
- BitTorrent schafft es, durch die ausgeklügelten Kooperations-Entscheidungs-Algorithmen, das *Leech*-Problem wirksam zu bekämpfen.
- BitTorrent erzielt damit regelmäßig Download-Raten von mehreren 100KB/s für eine einzelne Datei und ist damit wesentlich schneller als andere populäre Filesharing-Systeme.
- Anders als die meisten Filesharing-Systeme unterstützt Bittorrent das Verteilen nicht nur einzelne Dateien, sondern ganzer Verzeichnis-Bäume.

Schlussfolgerung

Mit BitTorrent ist ein effizienter Distributions-Weg für große Dateien entstanden. Die Client-Server-Struktur des Systems stört so lange nicht, so lange niemand dagegen ist, dass die freigegebenen Dateien verteilt werden. Da der Veröffentlicher erhebliche Infrastruktur bieten muss, die über das einfache „in's Netzwerk senden“ hinaus gehen, ist die Nutzung von BitTorrent als Datei-Fundus unattraktiv.

8. Zusammenfassung

- Alle hier vorgestellten Filesharing-Systeme haben kein Konzept für eine Dienstgüte. Das heißt zum Beispiel, dass der Abschluss des Downloads in einer bestimmten Zeit garantiert werden könnte.
  - Jedoch ist die Dienstgüte der Verbreitung per BitTorrent nie ineffizienter als die Verbreitung per HTTP oder FTP. Das liegt daran, dass im einfachsten Fall der Veröffentlicher den *Origin* betreibt, so als würde er den HTTP- oder FTP-Server betreiben. Gibt es also keine weiteren Lieferanten, so bleibt immer noch der Origin als Lieferant übrig.
- Die meisten Filesharing-Systeme sind trotz der peer2peer-Nutzung angreifbar, da sie zentrale Instanzen haben.

8. Zusammenfassung

- Der Trend geht dahin, Dateien per DateiID zu identifizieren und Segmentweise Hash-Werte zu bilden.

9. Ausblick

Die hier vorgestellten Filesharing-Systeme sind nur einige aus den über 50 inzwischen verfügbaren Architekturen. Viele davon stehen jedoch in den Startlöchern und einige kommen nie aus ihnen heraus. Viele versuche die Defizite von bestehenden Filesharing-Systemen anzugehen:

Verbesserungs-Ideen

Mehrschicht-Hashing

Ein Angriffspunkt ist, zu einer DateiID die falschen Segment-Hash-Werte zu liefern. Ein so misinformierter Client wird immer wieder korrekte Segmente verwerfen und so nie zum Ende des Downloads kommen. Dies kann man verhindern, indem die DateiID ein Hash-Wert über die Liste der Segment-Hash-Werte ist, nicht über die Datei selbst.

Übertragbare Credits

Kooperatives Verhalten wird derzeit nur zwischen direkt untereinander bekannten Teilnehmern bevorteilt. Wünschenswert wäre es, wenn ein Teilnehmer A ein Guthaben von Teilnehmer B auf ein Konto von C bei A übertragen könnte, wenn B und C dies wünschen. Somit wären Guthaben handelbar, und ein kooperativer Teilnehmer könnte auch von indirekt bekannten Teilnehmern bevorteilt werde. Wichtig wäre hierbei (wie in der realen Wirtschaft auch), dass die Guthaben langsam verfallen, damit

- die Guthaben-Besitzer motiviert sind, diese auch auszunutzen,
- die Guthaben nicht in's Unendliche wachsen und
- damit Neueinsteiger (ohne Guthaben) nicht benachteiligt werden.

Jeder Teilnehmer wäre somit seine eigene kleine (Zentral-)Bank.

Verteilte Relationen

Will man einen großen Datenbestand haben, der dezentral gespeichert wird, aber von jedem Teilnehmer komplett erreichbar ist, dann kommt man nicht herum, die Relationen, die einem Filesharing-Datenbankschema zugrunde liegen, auf mehrere Rechner zu verteilen. Dabei müssen sich die von einzelnen Rechnern abgedeckten Bereiche der Relation mehrfach überdecken, um Ausfall-Sicherheit zu erreichen. Diese Idee wird zum Beispiel vom Filesharing-System „Overnet“ verfolgt, welches jedoch leider proprietär ist.

10. Literatur

- Stephanos Androutsellis-Theotokis: „A Survey of Peer-to-Peer File Sharing Technologies. http://www.eltrun.gr/whitepapers/p2p_2002.pdf
- Filesharing-Geschichte allgemein: http://kop.fact.co.uk/DIVE/cd/text/p2p.html
- Filesharing allgemein: http://www.sockenseite.de/datentausch-minifaq.html
- Napster-Protokoll: http://opennap.sourceforge.net/napster.txt
- Napster-Geschichte: http://en.wikipedia.org/wiki/Napster
- Gnutella-Protokoll 0.4: http://rfc-gnutella.sourceforge.net/developer/stable/index.html
- Gnutella-Netzwerk-Visualisierung: http://www.cs.berkeley.edu/~pingster/viz/ieee.html http://www.ics.uci.edu/~danyelf/projects/gtv.html
- GWebCache-Spezifikation: http://www.gnucleus.com/gwebcache/newgwc.html
- FastTrack-Protokoll-Geschichte http://en.wikipedia.org/wiki/FastTrack
- eDonkey Protokoll: http://prdownloads.sourceforge.net/pdonkey/eDonkey-protocol-0.6.1.html http://www.giac.org/practical/GCIH/Ian_Gosling_GCIH.pdf http://www.w3seek.de/opendonkey/
- Beschreibung des parallelen, kooperativen Downloads von eDonkey http://www.edonkey2000.com/documentation/mftp.html
- BitTorrent-Protokoll http://bitconjurer.org/BitTorrent/protocol.html

Bildnachweis

- http://images.roxio.com/en/marcom/napster/kittybug_CMYK.zip
- http://oakspan.com/images/Gnutella-Logo.pdf
- http://www.kazaa.com/ugraphy.org/atlas/gnucleus_graph_large.gif
- http://static.designheavs/images/h_logo.gif
- http://unthesis.web.aplus.net/xmule_newsite_logo2.png
- http://ed2k-gtk-gui.sourceforge.net/img/menu/donkey.png
- http://www.cybergeoen.com/data/174/1workstation45.jpg
- http://static.designheaven.com/data/174/1workstation55.jpg
- http://static.designheaven.com/data/174/1workstation65.jpg
- http://static.designheaven.com/data/174/1mouse152.jpg
- http://static.designheaven.com/data/157/1trebleclef1.jpg
- http://www.edonkey2000.com/images/how2.gif
- http://www.emule-project.net/images/logo.gif
- http://www.emule-project.net/images/emule.gif
- http://www.emule.de/pics/network.gif
- http://f.scarywater.net/bram/central.png
- http://f.scarywater.net/bram/torrent.png

Weitere Quellen

- http://www.emule-project.net/faq/

v16