Darmstadt University of Technology

Peer-to-Peer Tauschbörsen – Eine Protokollübersicht

Dipl.-Wirtsch.-Ing. Oliver Heckmann, Dr. Jens Schmitt, Prof. Dr. Ralf Steinmetz

KOM Technical Report 06/2002
Version 1.0
November 2002

Last major update 12.11.02
Last minor update 19.12.02

E-Mail: {heckmann, schmitt, steinmetz}@kom.tu-darmstadt.de

Multimedia Communications (KOM)

Department of Electrical Engineering & Information Technology
& Department of Computer Science
Merckstraße 25 • D-64283 Darmstadt • Germany

Phone: +49 6151 166150
Fax: +49 6151 166152
Email: info@KOM.tu-darmstadt.de
URL: http://www.kom.e-technik.tu-darmstadt.de/

Zusammenfassung

Dieser Artikel erläutert die technischen Hintergründe und die Funktionsweise der bekanntesten Peer-to-Peer Tauschbörsen. Die erste große Tauschbörse, Napster, wird kurz dargestellt. Danach wird im Detail das eDonkey-Protokoll präsentiert, das besonders zum Tausch von Videos verwendet wird. Die bei eDonkey verwendeten Prinzipien sind typisch auch für andere aktuell erfolgreiche Tauschbörsen. Nicht vernachlässigen darf man Gnutella, die erste größere P2P Tauschbörse, welche ohne jegliche Server auskam. Gnutella wird samt den verschiedenen Problemen, mit denen es zu kämpfen hat, und den entsprechenden Lösungsmaßnahmen vorgestellt. Anschließend wird das momentan am stärksten verbreitete P2P Protokoll, das Fasttrack Protokoll, erläutert. Zu den genannten Protokollen werden außerdem diverse Messergebnisse präsentiert.

Vergleicht man die Entwicklungsrichtung der aktuellen Protokolle, so erkennt man viele Parallelen. Moderne leistungsfähige P2P Tauschbörsen verwenden die gleichen Prinzipien:

- Dateihashes zur Identifikation von identischen Dateien, was den gleichzeitigen Download von mehreren Quellen und die Fortsetzung unterbrochener Downloads erleichtert.
- Supernodes oder explizite Server zur effizienten Suche und Mechanismen, die die Konnektivität trotz Ausfalls mehrerer dieser Knoten sicherstellen.

Wir zeigen auch, warum bislang - von wenig erfolgreichen Spezialanwendungen wie das ebenfalls vorgestellte Freenet und Filetopia abgesehen - die Anonymität der Tauschbörsenteilnehmer nur mangelhaft geschützt ist. P2P ist ein aktuelles und interessantes Forschungsgebiet, deshalb wird am Ende des Artikels auf momentane technische Forschungsschwerpunkte eingegangen.

Abstract (english)

This article sheds light on the technical background as well as the detailed working of popular peer-to-peer file sharing systems. The first large-scale peer-to-peer file sharing system, Napster, is reviewed. Henceforth, the eDonkey protocol, which is very popular for sharing videos, is presented in detail. The principles and mechanisms that lie at the heart of eDonkey are typical for the latest, successful peer-to-peer file sharing systems. Of course, an overview of peer-to-peer file sharing systems would not be complete without Gnutella, the first “real” peer-to-peer system without any server elements. The protocol details as well as the many problems with which Gnutella initially had to cope are described. Next, the currently most popular peer-to-peer protocol, the Fasttrack protocol, is presented. Along with all these protocols, measurement results from experiments we conducted with the different systems are reported.

Comparing the evolution of the different systems several parallels become evident. Modern, well-performing systems apply the same principles and mechanisms:

- hashes to identify and locate identical files in order to allow for simultaneous downloads from several locations as well as continuation of interrupted downloads,
- so-called supernodes or, more explicitly, servers to enable efficient searching as well as mechanisms to increase fault-tolerance in case of failures of these nodes.

In addition, we discuss why - apart from notable exceptions like Freenet und Filetopia - so far the anonymity of participants in peer-to-peer file sharing systems has been neglected and why this will become important in the future. Peer-to-peer is an interesting and timely research field, hence we review some of the most interesting work in this area at the end of the article.

1 Einführung

Peer-to-Peer Tauschbörsen, die zum - häufig urheberrechtlich problematischen - Austausch von Liedern, Filmen, Büchern und anderen Dateien verwendet werden, haben in den letzten Jahren mehr und mehr Aufmerksamkeit in Presse, Industrie, Justiz und Forschung bekommen. Wie funktionieren diese Tauschbörsen, die angeblich die Musikindustrie in den Ruin treiben und der Filmindustrie gigantischen Schaden zufügen?

In diesem Artikel beschreiben wir die wichtigsten Peer-to-Peer (kurz P2P) Filesharing-Protokolle von einer technischen Perspektive, zeigen, wie es um die Anonymität bei diesen Tauschbörsen bestellt ist und geben einen Überblick über aktuelle Trends in der Forschung.

Wir beginnen mit Napster, der ältesten und in der allgemeinen Öffentlichkeit berühmtesten P2P-Tauschbörse, die heute allerdings nur noch ein Schattendasein fristet. Danach erläutern

wir, wie das in der “Szene” zum Tauschen von Videos sehr beliebte Protokoll, eDonkey 2000, funktioniert. Im Anschluss stellen wir das dezentral arbeitende Gnutella-Protokoll mit den ihm eigenen Problemen vor und beschreiben, wie man dieser Probleme Herr werden kann. Danach beschreiben wir das Protokoll der Firma Fasttrack, es ist die Grundlage für die momentan am weitesten verbreitete P2P Tauschbörse Kazaa. Der Artikel endet mit einer Beurteilung der Anonymität in P2P Tauschbörsen und Maßnahmen zur Erhöhung der Anonymität sowie einem kurzen Überblick über aktuelle technische Forschungsrichtungen im Bereich P2P Filesharing.

2 Napster – Aller Anfang ist schön, aber auch schwer

Traditionelle Client-Server Systeme sind asymmetrisch und bestehen aus einem Server, der eine sehr lange Zeit und mit hoher Zuverlässigkeit läuft und mehreren Clients, die auf die Dienste des Servers zugreifen. Auch das Internet wird, seitdem es mit dem WWW seinen großen Durchbruch erzielte, von dem Client-Server Prinzip dominiert: Clients greifen via Webbrowser auf Web- und FTP-Server zu, um verschiedene Dateien/Dokumente herunterzuladen und sich lokal anzuschauen. P2P-Anwendungen reduzieren die Asymmetrie zwischen Clients und Servern: Dateien werden vollständig zwischen zwei Clients ausgetauscht, Server sind je nach P2P-Anwendung gar nicht mehr nötig (Gnutella), haben nur noch eine eingeschränkte Funktion (Napster, eDonkey) zur effizienteren Suche der Dateien oder sind als solche gar nicht mehr erkennbar, da vollständig in die Clientanwendung integriert (Kazaa, aktuelle Entwicklungen bei Gnutella).

Napster (damals unter http://www.napster.com/) wurde im Mai 1999 entwickelt und war die erste große P2P Tauschbörse; Napster erlaubte es einem rapide wachsendem Benutzerkreis, Musikdateien (primär im MP3-Format) auszutauschen. Bei dem überwiegenden Teil der ausgetauschten Dateien wurden die Urheberrechte der Künstler und die Rechte der Musikindustrie ignoriert, was schließlich zu Klagen und zur “Kastrierung” des Dienstes sowie zur Insolvenz des Betreibers führte.

Das P2P-Prinzip war bei Napster noch nicht sehr stark ausgeprägt, so besaß das Napster-Netz noch eine sehr stark zentralisierte Topologie (Bild 1). Das Napster-Protokoll benutzt zur Suche von Dateien das Client-Server Prinzip und zum Austausch (Herunterladen) der Dateien das P2P-Prinzip. Die Benutzer von Napster installieren die Napster-Anwendung auf ihrem Rechner und wählen einen eindeutigen Benutzernamen. Die Anwendung registriert zunächst die von dem Benutzer zum Tausch angebotenen Musikdateien und übermittelt diese Liste an den zentralen Napster-Server. Sucht ein Benutzer nach einem bestimmten Musikstück, so wird die

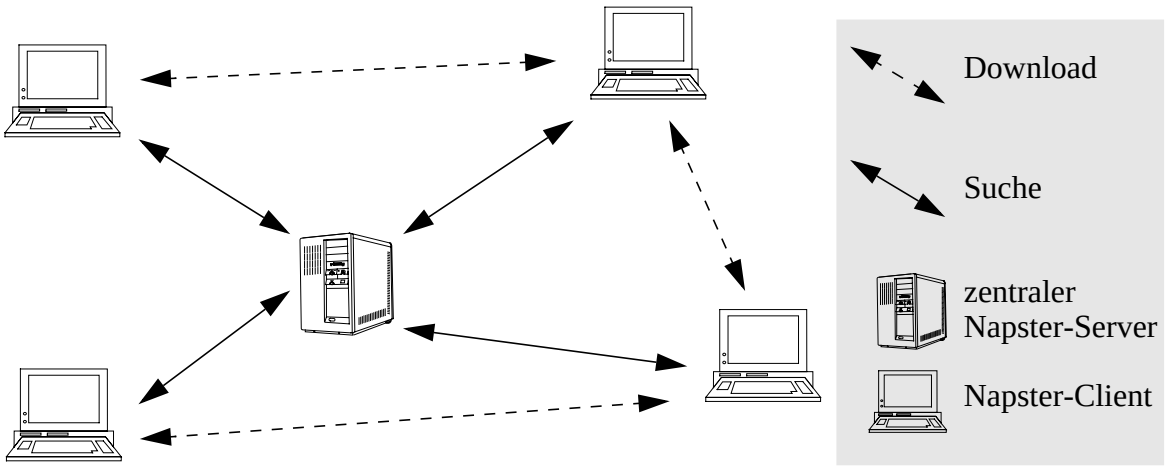


Bild 1 Aufbau des Napster-Netzwerkes

Suchanfrage an den zentralen Server übergeben, der mit einer Liste von Anbietern des genannten Musikstücks antwortet. Der Download der eigentlichen Datei findet danach ohne Beteiligung des Servers direkt zwischen den Clients/Peers (also Peer-to-Peer) statt. Benutzt wurde dieses Prinzip, weil der Betreiber des Napster-Servers (letztlich fälschlicherweise) darauf spekulierte, sich erfolgreich gegen die Klagen der Musikindustrie wehren zu können, wenn auf seinem Server keine Musikstücke zwischengespeichert werden und sein Server nicht direkt am Austausch der Musikstücke beteiligt ist.

Die Verwendung eines zentralen Servers war die Schwachstelle des Dienstes, da sie zu einer "Kastrierung" des Dienstes und zur Abwanderung der Benutzer führte, nachdem die Musikindustrie die Installation von Filtern auf dem zentralen Server erzwang, die die Suche nach kopiergeschützten Inhalten stoppten.

Mittlerweile ist die Firma Napster insolvent, das Napster-Protokoll wurde geknackt und lebt weiter in dem Open Source Projekt OpenNap (http://opennap.sourceforge.net/). Die meisten Benutzer sind zu anderen, moderneren Tauschbörsen abgewandert. Weitere Details über das Protokoll finden sich unter [Ope00].

3 eDonkey – P2P Filesharing wird erwachsen

eDonkey 2000 (http://www.edonkey2000.com) ist einer der Napster-Nachfolger und besonders beliebt zum Austausch von Filmen und anderer großer Dateien. Im eDonkey-Netz finden sich fast alle auf DVD erhältlichen Filme und eine Vielzahl noch gar nicht auf DVD veröffentlichter aktueller Kinofilme.

Eine Besonderheit von eDonkey ist, dass Dateien anhand eines MD4 ähnlichen Hashwertes [Riv90] und ihrer Dateilänge und nicht wie z.B. bei Napster oder bei Gnutella über ihren Datei-

namen identifiziert werden. Dies hat den Vorteil, dass identische Dateien auch identifiziert werden, wenn sie unterschiedliche Namen wie z.B. "Lord of the Rings.avi" oder "Lord of the Rings (gute Qualität).avi" etc. haben. Hierdurch wird eine Besonderheit des eDonkey-Protokolls unterstützt, nämlich dass eine Datei gleichzeitig von einer Vielzahl von Quellen heruntergeladen werden kann. Hiervon verspricht man sich höhere Netto-Downloadraten. Außerdem ist es hierdurch leichter, einen Download von einer anderen Quelle fortzusetzen, nachdem die ursprüngliche Quelle offline gegangen ist, etwas das gerade beim Herunterladen größerer Dateien häufiger vorkommen kann. Wir haben bei unseren Messungen viele Filme mit 600-800 MB heruntergeladen; der Download eines Films kann durchaus mehrere Tage dauern, in diesem Zeitraum werden die Quellen, von denen der Film geladen wird, häufiger gewechselt. Das eDonkey zugrunde liegende Protokoll wird aus diesem Grund auch Multisource File Transfer Protocol (MFTP) genannt.

Im eDonkey-Protokoll werden ähnlich wie bei Napster Client und Server unterschieden. Die Server-Software ist allerdings frei verfügbar, womit potenziell jeder Benutzer einen eDonkey-Server anbieten kann. Das Ausschalten aller Server durch Gerichtsentscheide wird damit praktisch unmöglich. Das Protokoll unterstützt ferner explizit die Möglichkeit, z.B. Suchanfragen an mehr als einen Server zu richten und interne Daten zwischen mehreren Servern auszutauschen.

Die Server stellen den Suchdienst zur Verfügung, während die Clients die Dateien speichern. Downloads finden ausschließlich direkt Client zu Client statt; auf den Servern werden keine Inhalte gespeichert oder zur Verfügung gestellt, über sie läuft auch kein Downloadverkehr.

In der Beschreibung des Protokolls unterscheiden wir jetzt zwischen dem Server-Server, Client-Server und Client-Client Verkehr. Wir wollen dieses Protokoll etwas genauer beschreiben, zum einen, weil dieses Protokoll in wissenschaftlichen Untersuchungen bislang vernachlässigt wurde, und zum anderen, weil dieses Protokoll sehr erfolgreich ist und deshalb mittlerweile auch in anderen Tauschbörsen (z.B. Overnet, http://www.overnet.com/) verwendet wird. Unsere Erkenntnisse stammen aus Mitschnitten des Verkehrs zwischen Clients und Servern mit den Tools TCPDump [Dump] und TCPFlow [Elson]. Darauf basierend haben wir auch einen eigenen eDonkey-Server geschrieben, um Messungen im eDonkey-Netz durchführen zu können.

MFTP benutzt TCP (Transmission Control Protocol) und UDP (User Datagram Protocol) als Transportprotokolle, bei TCP werden virtuelle Verbindungen zwischen den beiden Kommunikationsteilnehmern aufgebaut, über die in beide Richtungen Daten ausgetauscht werden. TCP sorgt automatisch dafür, dass verlorene Pakete erneut versendet werden und stellt die korrekte

Reihenfolge beim Empfang sicher. Die Fluss-/Überlaststeuerung von TCP sorgt zudem dafür, dass weder ein langsamer Empfänger noch ein überlastetes Netzwerk mit Daten überschwemmt werden. UDP stellt nur einen einfachen Datagrammdienst zur Verfügung, über den in eine Richtung Pakete verschickt werden können, ohne dass der korrekte Empfang oder ein Empfang in der richtigen Reihenfolge sichergestellt sind.

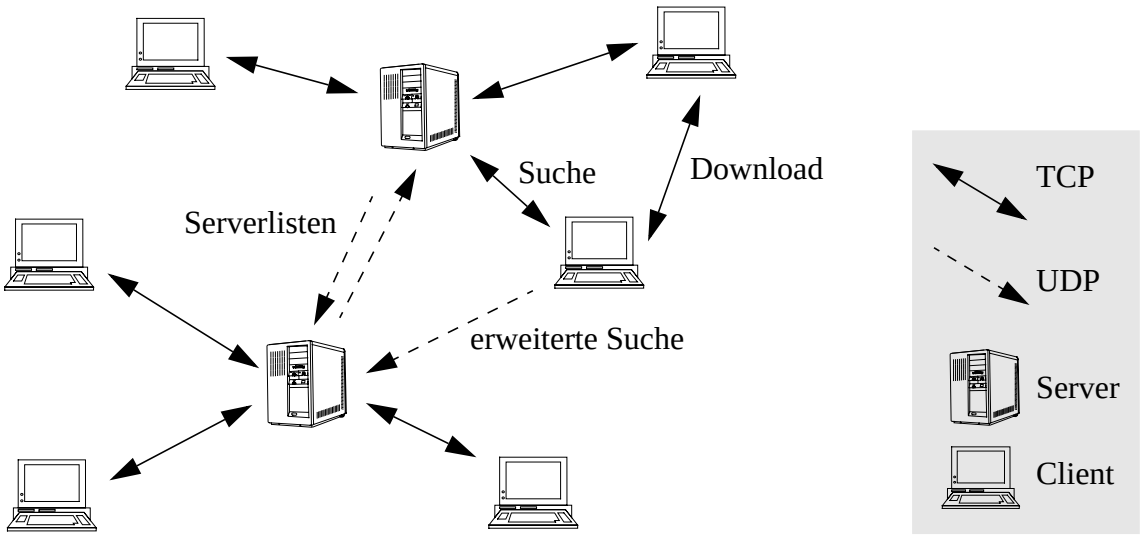


Bild 2 Aufbau des eDonkey-Netzwerkes

Das erste Byte jeder eDonkey-Nachricht besteht einleitend aus dem Byte 0xE3, bei TCP-Nachrichten gefolgt von der Größe der Nachricht in Byte als vorzeichenloser Integerwert; bei UDP-Nachrichten ist dies nicht nötig, da die Nachrichtenlänge durch UDP bestimmt wird. Anschließend folgt die Nachrichten-ID, die den Typ der Nachricht beschreibt. Die ID der Nachricht wird im Folgenden immer hexadezimal in Klammern angegeben.

3.1 Server-Server Verkehr

Der Server-Server Verkehr ist relativ einfach. Ein Server besitzt eine Liste mit einer größeren Zahl anderer eDonkey-Server. Nach der Installation eines neuen Servers können weitere Server per Hand hinzugefügt werden oder es kann eine Serverliste installiert werden, die eine Vielzahl von aktiven Servern erhält und die im WWW verfügbar sind (z.B. auf der Homepage des Projektes, http://www.eDonkey2000.com, oder auf verschiedenen speziellen Seiten wie http://ocb-maurice.dyndns.org). Wird der Server das erste Mal gestartet, schickt er als erstes eine "Server Announce" Nachricht (ID 0xA0) via UDP an die Server seiner internen Serverliste. Mit dieser Nachricht teilt er den anderen Servern seine Verfügbarkeit mit und diese nehmen ihn in ihre Serverliste auf. Veraltete Einträge in den Listen werden spätestens nach ein paar Tagen gelöscht.

Während des Betriebs tauschen die Server untereinander ihre Serverlisten aus. Hierzu werden UDP-Nachrichten (ID 0xA1) verwendet. Ein neuer Server hat also nach einiger Zeit einen aktuellen Überblick über die gerade funktionsfähigen Server, auch wenn seine ursprüngliche Serverliste nur wenige funktionierende Server enthielt. Dies macht das Protokoll relativ robust gegenüber dem Ausfall von Servern, insbesondere im Vergleich zu Napster, wo der Name des zentralen Napster-Servers hardcodiert war.

3.2 Client-Server Verkehr

Ein so gestarteter Server wartet nun darauf, dass Clients zu ihm Verbindung aufnehmen; der Client-Server Verkehr sieht wie folgt aus:
Wird der Client gestartet, so berechnet er zunächst den MD4 Hashwert aller Dateien, die er anbietet. eDonkey geht davon aus, dass zwei Dateien identisch sind, wenn ihr Hashwert und ihre Dateigröße identisch sind, der Dateiname spielt keine Rolle. Der Client versucht jetzt eine TCP-Verbindung zu einem Server herzustellen, auch er führt hierzu eine interne Serverliste. Die Server haben üblicherweise ein Limit an Verbindungen, die sie akzeptieren, deshalb muss es der Client ggf. bei mehreren Servern probieren, bis ihm der Verbindungsaufbau gelingt; dieser Vorgang kann durchaus einige Minuten dauern. Über die neu aufgebaute Verbindung schickt der Client an den Server zunächst eine "Hello" Nachricht (ID 0x01). Diese enthält unter anderem die IP-Adresse und den Port des Client, den (frei wählbaren) Benutzernamen und die Version der Clientsoftware. Der Server antwortet mit einer "Server Stats" Nachricht (ID 0x41), die den Servernamen und einen beschreibenden String enthält. Beides kann durch den Betreiber des Servers gesetzt werden und typischerweise finden sich hier stolze Angaben des Betreibers wie "Pentium 4 1,8 GHz - 512 MB RAM" etc. Der Server schickt dem Client außerdem eine Begrüßungsnachricht, die beim Client auf dem Bildschirm angezeigt wird. In diesen Begrüßungsnachrichten steht häufig das - wohl nicht wirklich ernst gemeinte - Verbot, illegale Inhalte zu tauschen, sowie die Aufforderung, heruntergeladene Inhalte eine Zeit lang (z.B. 3 Tage) zu "sharen", also anderen Benutzern zum Herunterladen anzubieten. Für jede Zeile der Nachricht wird die "Write String" Nachricht (ID 0x38) verwendet. In unregelmäßigen Abständen schickt der Server seinen Clients die "User/Files" Nachricht (ID 0x34), die die Anzahl der Benutzer, die mit dem Server verbunden sind, und die von ihnen angebotenen Dateien enthält. Typische über eine DSL Leitung betriebene Server haben meistens ein Limit von 800 bis 2000 Benutzern, bei gut besuchten Servern ist dieses Limit i.d.R. ausgelastet. Im Schnitt werden bei 1000 Benutzern ungefähr 50.000-70.000 Dateien angeboten. Es gibt aber auch Server, die über eine hervorragende Internetanbindung verfügen und 20.000 Benutzer und etwa 600.000 Dateien verwalten[1].

Nachdem eine Verbindung aufgebaut wurde und der Client die Hashwerte seiner Dateien berechnet hat, teilt er dem Server alle Dateien mit, die er anbietet. Hierzu wird der gleiche Nachrichtentyp "Shares/Search Result" (ID 0x33) verwendet, der auch später bei den Antworten auf eine Textsuche verwendet wird (siehe unten). Der Server führt Buch über die Dateien, die seine Clients anbieten.

Der Server schickt dem Client auch regelmäßig seine Serverliste zu (ID 0x32 über TCP). Damit ist auch der Client ständig über aktuelle eDonkey-Server informiert und findet beim Ausfall eines Servers schnell einen neuen.

Will der Benutzer eine Datei suchen, so sucht er in der Regel erst mit einer Textsuche nach dieser Datei. Hierbei wird der Suchstring, den der Benutzer eingibt (z.B. "Herr der Ringe") mit einer "Text Search" Nachricht (ID 0x16) an den Server geschickt. Bei der Suche kann auch der Filetyp spezifiziert werden (Musik, Video, Software, Dokument). Der Server antwortet mit einer "Shares/Search Result" Nachricht (ID 0x33), die alle Dateien enthält, deren Dateiname die Wörter der Suchnachricht enthält und die ggf. vom geforderten Dateityp sind. Für jede Datei werden der Hashwert, der Dateityp, der Dateiname, die Dateigröße, die Anzahl der Benutzer, die die Datei anbieten, und dateitypabhängige Informationen (wie z.B. bei Liedern Länge, Künstler, Album, und Bitrate der Kodierung) angegeben. Der Benutzer sieht nun die Suchergebnisse. Sind diese ihm nicht ausreichend, kann er eine erweiterte Suche bei anderen Servern einleiten. Hierbei wird der Suchstring mit der "UDP Text Search" Nachricht (ID 0x98) jeweils an einen weiteren Server geschickt. Es wird UDP verwendet, damit keine aufwändige TCP-Verbindung zu diesen Servern hergestellt werden muss und weil es nicht kritisch ist, wenn diese Nachrichten verloren gehen. Diese Server bearbeiten die Suchnachricht und schicken ebenfalls via UDP ihre Suchantwort zurück; die Nachricht "UDP Search Result" (ID 0x99) entspricht der "Shares/Search Result" Nachricht, allerdings wird für jede einzelne passende Datei eine Antwortnachricht generiert. Wird die erweiterte Suche zu stark genutzt, kommt leicht eine große Menge an unkontrolliertem UDP Verkehr zustande.

Der Benutzer sieht jetzt die Suchresultate. Für die oben genannte Suche "Herr der Ringe" könnte er z.B. als Suchantwort 4 Dateien "Herr der Ringe.avi", "Herr der Ringe (DVD Rip von Mister X).avi", "Herr_der_Ringe.jpg" und "Soundtrack - Herr der Ringe.zip" finden (zugegebenermaßen wird ein Benutzer bei einem populärem Thema wie “Herr der Ringe” er-

1. Bei den größeren Servern liegt der Schnitt Dateien / Client deutlich unter dem von gut besuchten kleineren Servern. Die größeren Server können sich das auch “erlauben”, da sie wegen der höheren Benutzerzahl dennoch eine große Zahl angebotener Dateien erreichen.

heblich mehr Resultate präsentiert bekommen, aber wir wollen das Beispiel einfach halten. Wir haben in einem kleinen Test z.B. 216 unterschiedliche (!) Dateien für den Begriff "Herr der Ringe" gefunden). Die ersten beiden Dateien haben den gleichen Hashwert und die gleiche Dateigröße, damit geht eDonkey davon aus, dass es sich um Dateien mit dem gleichen Inhalt handelt. Damit ist es egal, ob der Benutzer jetzt die erste oder zweite Datei zum Download auswählt, in beiden Fällen führt eDonkey jetzt vor dem eigentlichen Download eine Suche nach konkreten Anbietern der Datei durch, wobei die Datei nur noch über den Hashwert und die Größe identifiziert wird. Diese Suche wird mit der "Query Sources" Nachricht (ID 0x19) durchgeführt, auf die der Server mit "Return Download Sources" (ID 0x42) antwortet. Die Antwort enthält eine Liste der IP-Adressen und Ports der Clients, die die Datei komplett oder in Fragmenten anbieten.

Neben der oben geschilderten Textsuche hat sich mittlerweile eine zweite Möglichkeit etabliert, nach angebotenen Dateien zu suchen. Im eDonkey-Netz wurden früher häufig auch so genannte "Fakes" (Fälschungen) ausgetauscht. Dies sind Dateien mit einem absichtlich irreführenden Dateinamen. So hat man scheinbar noch nicht im Kino gestartete Filme gefunden, um nach einem langen Download festzustellen, dass es sich um eine andere Datei mit falschem Dateinamen oder schlicht um eine kaputte Datei handelt. Bei Musikstücken kommt es mittlerweile häufiger vor, dass es sich nur um unvollständige Lieder handelt, die absichtlich von der Plattenfirma verbreitet werden. Um sich den umfangreichen und ärgerlichen Download von solchen Fälschungen zu sparen, entstanden schnell Webseiten (z.B. http://www.sharereactor.com), auf denen die Namen und Hashwerte der Fälschungen und auch der echten Dateien zugänglich sind. Mittlerweile haben sich diese Seiten weiterentwickelt und indizieren große Teile des eDonkey Dateiangebots (z.B. http://jigle.com), so dass viele Benutzer gar nicht mehr den Suchmechanismus von eDonkey benutzen, sondern sich ausschließlich auf die Suchmechanismen dieser Seiten verlassen. Diese Seiten refinanzieren sich wiederum durch Werbeeinnahmen. Mit einem Klick auf einen Link der Art “ed2k://|file|filename|filesize|MD4Hash|” kann man direkt den Download der damit bezeichneten Datei starten. Der eDonkey-Client führt dann direkt die Suche nach Quellen der Datei mit dem "Query Sources"-Mechanismus durch.

3.3 Client-Client Verkehr

Zum Download der Datei findet jetzt die Client-Client Kommunikation statt.

Ein Client, der eine bestimmte Datei herunterladen will, öffnet zu mehreren der Clients, die er mit dem “Query Sources” Mechanismus gefunden hat, TCP Verbindungen und lädt über sie verschiedene Teile (bis zu 9.28 MB große so genannte “Chunks”) herunter, bis er die Datei

komplett zusammensetzen kann. Die Teile, die er bereits heruntergeladen hat, bietet er sofort auch wieder weiteren Clients an. Sehr beliebte Dateien verbreiten sich somit rasant, da nicht auf einen abgeschlossenen Transfer gewartet werden muss, bis die Dateien weitergegeben werden.
Beim Client-Client Protokoll werden Kontrollnachrichten und Daten über eine geöffnete TCP-Verbindungen zwischen 2 Clients ausgetauscht.
Client A will eine Datei von Client B herunterladen. Dazu baut Client A eine TCP-Verbindung zu B auf und identifiziert sich zunächst mit einer “Hello” Nachricht (0x01), in der er seinen Benutzernamen, Clientversion etc. mitteilt. Diese Nachricht wird von B mit einer “Hello Back” Nachricht (0x4C) quittiert, die ähnliche Informationen enthält.
Mit einer “Query File” Nachricht (0x58) kann A nach der gewünschten Datei fragen, die Nachricht enthält den Hashwert der Datei, die Antwort (0x59) von B darauf enthält den lokalen Dateinamen - oder die Meldung, dass B die Datei nicht besitzt (0x48). Jeder Client hat nur eine bestimmte Menge an “Upload-Slots”; um eine Datei herunterladen zu können, muss man einen solchen Slot zugeteilt bekommen. Auf die Anfrage nach einem Slot von A (0x54) antwortet B, ob ein Upload-Slot (0x55) verfügbar ist oder nicht (0x57). B kann somit den ausgehenden Datenverkehr beschränken.
A kann nach einem bestimmten Ausschnitt aus der gewünschten Datei fragen (0x47), die gewünschten Daten werden dann übertragen (0x46) und mit einer Endenachricht quittiert (0x49), sofern der Downloadvorgang von A nicht unterbrochen wird (0x56). Da der Hashwert jedes 9.28 MB Chunks aus dem Gesamthash hervorgeht, kann A überprüfen, ob er die korrekten Daten empfangen hat bzw. ggf. die Daten bei einem anderen Client nachfragen, wenn er von einem Client fehlerhafte Daten bezieht. Das Protokoll ist damit relativ robust gegenüber Störern.
Sitzt B hinter einer Firewall, die eingehende aber nicht ausgehende TCP Verbindungen blokkiert - was bei Firewalls eine übliche Konfiguration darstellt - , so kann A keine TCP Verbindung zu B öffnen. Auch dieser Fall wird von dem eDonkey-Protokoll explizit berücksichtigt. Wenn sich B bei seinem Server anmeldet, überprüft der Server, ob er eine TCP-Verbindung zu B öffnen kann oder nicht. Wenn nicht, sitzt B vermutlich hinter einer Firewall. Beim Anmeldeprozess teilt der Server B eine ID zu, sitzt B hinter einer Firewall, so hat diese ID einen sehr niedrigen Wert, ansonsten bekommt B einen sehr hohen Wert zugeteilt (Größenordnung 10^9). Sitzt B hinter einer Firewall, so kann A den Server von B auffordern, B zu bitten, eine TCP Verbindung zu A aufzubauen.
Der Mechanismus, über UDP-Nachrichten bei verschiedenen Servern zu suchen oder nach Quellen zu fragen, erlaubt es sehr effektiv, schnell große Teile des eDonkey-Netzes zu durch-

forsten. Auf der anderen Seite sind hiermit allerdings auch Probleme aus Netzbetreibersicht verbunden, die nicht verschwiegen werden sollen. UDP-Nachrichten können verschickt werden, auch ohne dass die Empfängeranwendung oder gar der Empfängerrechner aktiv sind (bei TCP würde der Verbindungsaufbau scheitern). Außerdem passiert es aufgrund der mangelnden Fluss- und Überlastkontrolle schnell, dass ein Netz oder ein Rechner mit UDP-Nachrichten überlastet wird, ohne dass der Sender dies bemerkt. Der Verkehr eines eDonkey-Servers ist zum überwiegenden Teil UDP Verkehr; unsere Messungen zeigen ein Verhältnis UDP:TCP von 9:1. Ein Server ist leicht mit diesem Verkehr überlastet und der Verkehr wird trotz der Überlast nicht zurückgehen. Bei unseren Messungen fiel uns außerdem auf, dass 12 Stunden, nachdem unser eDonkey-Server gestoppt wurde, an dessen alte Adresse noch eine stattliche Zahl von UDP-Nachrichten geschickt wurden. Vor dem Hintergrund, dass die Absenderadressen von UDP-Nachrichten leicht gefälscht werden können, bietet dieser Mechanismus außerdem eine einfache Möglichkeit, die große Zahl von eDonkey-Servern, deren Adressen leicht zu finden sind, zu verteilten Denial-of-Service Attacken zu missbrauchen. Unserem Wissen nach wurde der Mechanismus hierzu allerdings bislang noch nicht ausgenutzt.

Wie wir gezeigt haben, ist das eDonkey-Protokoll in vielerlei Hinsicht mächtiger und robuster als das Napster-Protokoll. Alternative eDonkey-Clients gibt es unter http://www.emule-project.net/ und http://savannah.nongnu.org/projects/mldonkey/. Es gibt neuerdings auch eine serverlose Variante des eDonkey-Protokolls, Overnet (http://www.overnet.com). Wir beschreiben deshalb als nächstes Gnutella, das klassische Beispiel für ein serverloses und damit völlig dezentrales P2P Protokoll.

4 Gnutella – alle sind gleich

Das Gnutella-Protokoll ist wie erwähnt völlig dezentral und hatte daher anfangs mit vielen Performanceproblemen zu kämpfen. In der letzten Zeit hat es daher verschiedene Verbesserungen erfahren. Wir stellen zunächst das ursprüngliche Protokoll vor und beschreiben am Ende die mittlerweile vorgenommenen Verbesserungen.

Das Gnutella-Protokoll und der erste Client wurden im März 2000 entwickelt und wenige Stunden nach der Veröffentlichung bereits auf Betreiben der Hersteller- bzw. Mutterfirmen Nullsoft, AOL, Time Warner wieder zurückgezogen. Die kurze Zeit hatte allerdings gereicht, um die Software im Netz zu verteilen. Das Protokoll war schnell geknackt und veröffentlicht [Cli00] und seitdem gibt es eine Vielzahl von Gnutella-Clients (z.B. Limewire, Bearshare, Morpheus, XoloX[1]), die von einer Vielzahl von Firmen und Entwicklern weiterentwickelt werden. Wir

stellen im Folgenden die ursprüngliche Version des Protokolls aus dem Jahr 2000 vor (Version 0.4) und gehen am Ende auf mittlerweile vorgenommene Verbesserungen ein.

Das Gnutella-Netzwerk ist ein vollständig dezentrales Netzwerk, d.h. es kommt ohne jegliche zentrale Server aus. Stattdessen übernimmt jeder Knoten bei Gnutella gleichzeitig die Funktion eines Clients und eines Servers und wird als Anspielung darauf auch “Servent” genannt.

Das Gnutella-Netzwerk besteht aus den miteinander verbundenen Servents. Ein neuer Servent verbindet sich mit dem Gnutella-Netzwerk, indem er zunächst eine TCP-Verbindung mit mindestens einem Servent des Netzwerks aufbaut. Dazu muss er die IP-Adresse und den Port eines Servents kennen, der sich momentan im Netz befindet. Um eine solche Adresse herauszufinden, gibt es verschiedene Wege. Früher wurden Adressen in Chat-Kanälen und im Web veröffentlicht, mittlerweile haben sich so genannte “Host Caches” etabliert, die unter einem festen DNS Eintrag (z.B. gnutellahosts.com) erreichbar sind und die Adressen unzähliger Knoten kennen (nämlich die Knoten, die bei ihnen Adressen nachfragen).

Sobald eine TCP Verbindung steht, werden darüber die Gnutella-Nachrichten ausgetauscht. Das Gnutella-Protokoll besitzt einen Mechanismus (PING/PONG), der genutzt werden kann, um das Gnutella-Netz zu erkunden und die Adressen einer Vielzahl weiterer Knoten auch ohne Host Caches herauszufinden. Zu einer bestimmten Anzahl (z.B. 4) anderen Knoten wird eine TCP Verbindung hergestellt.

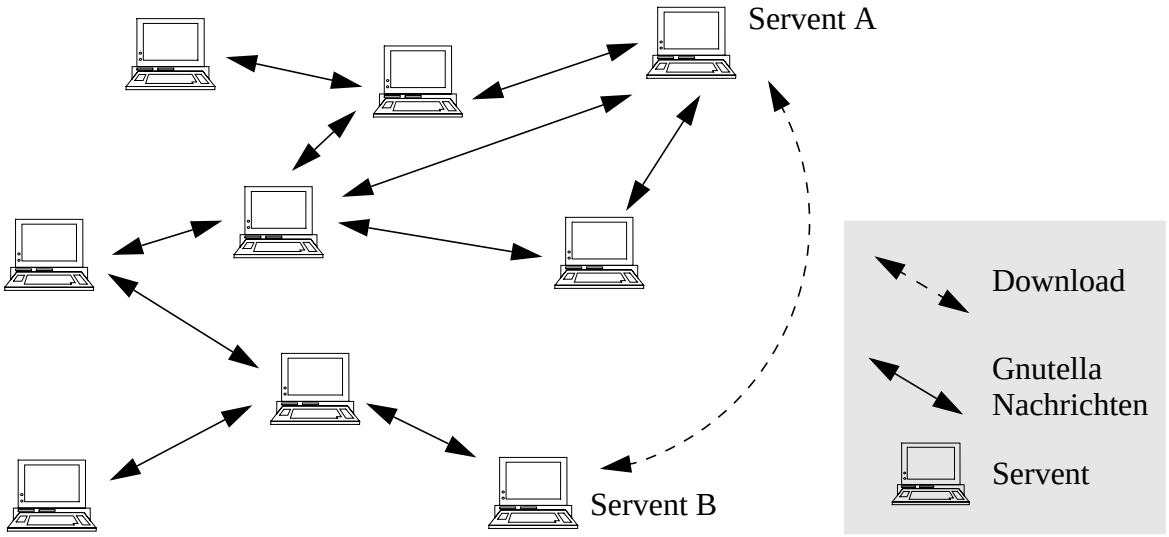


Bild 3 Aufbau des Gnutella-Netzwerkes

Die meisten Gnutella-Nachrichten werden gebroadcastet, d.h. ein Servent, der von einem anderen eine Nachricht erhält, schickt sie an alle anderen verbundenen Servents weiter. Erhält er die Nachricht ein zweites Mal, so verwirft er sie. Daraus folgt, dass ein Mechanismus benötigt wird,

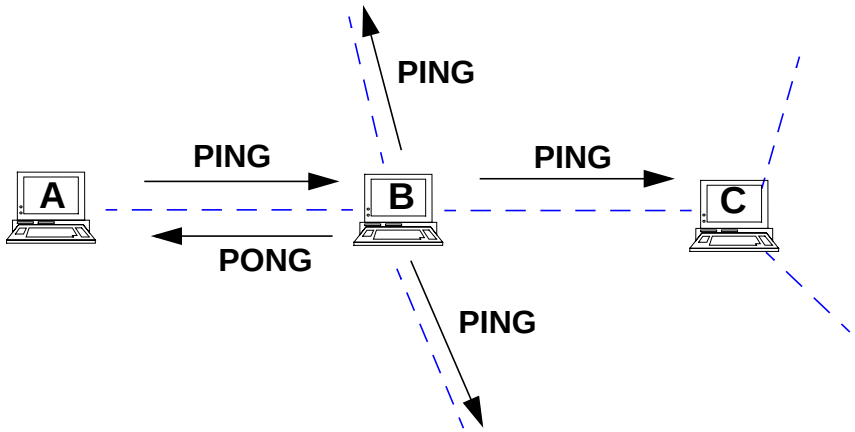
1. Siehe www.limewire.com, www.bearshare.com, www.musiccity.com, www.xolox.nl.

mit dem man eine bereits erhaltene Nachricht leicht identifizieren kann. Jede Gnutella-Nachricht hat im ihrem Header eine 16 Byte lange “Descriptor ID”. Diese wird mit einem Algorithmus erzeugt, der es extrem unwahrscheinlich macht, dass in einem gewissen Zeitraum zweimal die gleiche Descriptor ID erzeugt wird. Man kann demnach davon ausgehen, dass jede Nachricht über die Descriptor ID eindeutig identifiziert werden kann. Jeder Knoten speichert nun für eine gewisse Zeit, welche Descriptor IDs er weitergeleitet hat, taucht eine Nachricht mit derselben Descriptor ID erneut auf, so verwirft er sie. Damit eine Gnutella-Nachricht nicht unendlich lange durch das Netz schwirrt, enthält sie im Header zusätzlich ein TTL (Time-To-Live) Feld, das die Anzahl der Hops angibt, die die Nachricht maximal weitergeleitet werden soll, und ein Hops-Feld, in dem steht, wie häufig die Nachricht bereits weitergeleitet wurde.

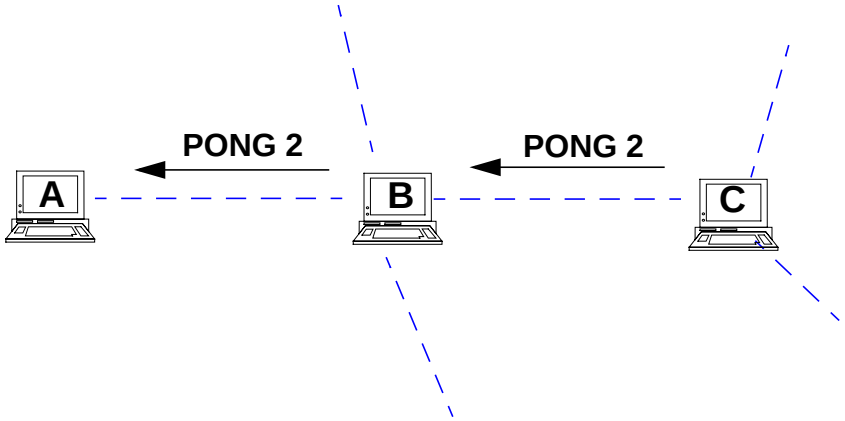
Es gibt 5 verschiedene Nachrichtentypen. Die PING- und PONG-Nachrichten dienen dem Erkunden und Kennenlernen des Gnutella-Netzes und sind aufgrund der völlig dezentralen Struktur des Protokolls nötig. QUERY und QUERYHIT dienen dem Durchsuchen des Netzes nach Dateien während PUSH nötig ist, um in bestimmten Fällen Firewalls zu umgehen.

Ein Servent verschickt PING-Nachrichten, um das Netz zu erkunden. Eine PING-Nachricht enthält keinerlei Daten (auch nicht die IP-Adresse des Servents, der sie losschickt) und wird mit dem oben beschriebenen Broadcast-Mechanismus durch das Gnutella-Netzwerk geleitet. Jeder Servent, der eine PING-Nachricht erhält, kann mit einer PONG-Nachricht antworten. Die PONG-Nachricht enthält die IP-Adresse und den Port eines Servents, der sich im Gnutella-Netz befindet, typischerweise ist es die Adresse des Servents, der die PONG-Nachricht erstellt. Zusätzlich enthält die Nachricht die Anzahl der Dateien und die Summe der Dateigrößen (in KByte) des Servents. Die PONG-Nachricht wird nicht gebroadcastet, sondern nur an den Servent zurückgeschickt, über den die PING-Nachricht eintraf. Die PONG-Nachricht bekommt ferner die gleiche Descriptor ID wie die PING-Nachricht, die sie beantwortet. Bild 4 verdeutlicht das Vorgehen an einem Beispiel.

Um nach Dateien zu suchen, broadcastet ein Servent eine QUERY-Nachricht, diese enthält einen Suchstring. Sie enthält nicht die IP-Adresse des suchenden Servents. Jeder Servent, der die Nachricht erhält, überprüft, ob er eine Datei anbietet, die dem Suchstring entspricht. Ist dies der Fall, antwortet der Servent mit einer QUERYHIT-Nachricht, der er die Descriptor ID der QUERY-Nachricht gibt. Da er die IP-Adresse des suchenden Servents nicht kennt, wird die QUERYHIT-Nachricht genauso wie eine PONG-Nachricht entlang der Route, die die korrespondierende QUERY-Nachricht nahm, zurückgeroutet.



1. **A verschickt eine PING-Nachricht an B.**
2. **B leitet die PING-Nachricht weiter an die mit ihm verbundenen Servents.**
3. **B antwortet auf die PING-Nachricht mit einer PONG-Nachricht nur an A.**



4. **C antwortet mit einer PONG-Nachricht (PONG 2). Die-Nachricht wird nur an B verschickt, weil die PING-Nachricht über B an C ging.**
5. **Anhand der Descriptor ID der Nachricht hat B gemerkt, dass es sich um eine Antwort auf den PING von A handelt. B leitet die Nachricht nur an A weiter.**

Bild 4 PING- und PONG-Nachrichten werden im Gnutella-Netz unterschiedlich geroutet.

Die QUERYHIT-Nachricht enthält IP-Adresse und Port des anbietenden Servents, die Geschwindigkeit seiner Internetanbindung und eine Liste von angebotenen Dateien, die dem Suchstring entsprechen, mit Dateiname und Größe. Moderne Gnutella-Clients senden noch weitere Metainformationen über die Datei, z.B. die Kodierungsrate von Audiodateien.

Um eine Datei herunterzuladen, baut der suchende Servent eine TCP Verbindung zu dem anbietenden Servent auf. Der Dateitransfer findet einfach mit HTTP (Hypertext Transfer Protocol, [FGM+99]) statt. Das ursprüngliche Gnutella-Protokoll unterstützt nicht das Herunterladen von mehreren Quellen gleichzeitig oder die Wiederaufnahme eines unterbrochenen Downloads von der gleichen oder einer anderen Quelle. Dies hat dazu geführt, dass im Jahr 2000 lediglich 10% aller Downloads erfolgreich waren [Lim01]. Moderne Clients unterstützen die Wiederaufnahme eines unterbrochenen Downloads und das Herunterladen von mehreren Clients gleichzeitig,

die Erfolgsquote ist daher erheblich größer. Ebenso wird mittlerweile auch eine Hashfunktion zur Identifikation identischer Dateien ähnlich wie bei eDonkey verwendet.

Falls der anbietende Servent A hinter einer Firewall sitzt, die die ankommende HTTP/TCP Verbindung zu ihm blockiert, so schickt der Servent B, der von A eine Datei herunterladen will, an A über das Gnutella-Netz (d.h. die bereits offenen Verbindungen) eine PUSH-Nachricht mit seiner IP-Adresse. Diese Nachricht ist eine Aufforderung an A, zu B eine HTTP/TCP Verbindung aufzubauen - typischerweise werden nämlich ausgehende HTTP/TCP Verbindungen nicht von Firewalls geblockt. Über die erstmal aufgebaute Verbindung kann dann die Datei übertragen werden.

In einer im Jahr 2000 durchgeführten Studie deckten Adar und Hubermann [AH00] ein erschreckendes, wenngleich wenig überraschendes Benutzerverhalten anhand von Messungen im Gnutella-Netz auf: 66% der aktiven Benutzer boten überhaupt keine und 73% lediglich 10 oder weniger Dateien zum Download an - eine digitale Form des Allmende-Problems. Diese “Free-Rider” waren gleichmäßig über die verschiedenen Top-Level Domains (com, edu, net, de, uk, usw.) verteilt. Zu dem Ergebnis, dass ca. 10% der Knoten 99% des Verkehrs in P2P-Netzen erzeugen, kam eine weitere Studie [SW02]. Wir haben Anfang Oktober 2002 ähnliche Messungen durchgeführt[1]. Das Free-Rider Problem hat deutlich abgenommen, vermutlich weil die Gnutella-Clients mittlerweile die Möglichkeit vorsehen, Verbindungen mit Free-Ridern abzulehnen. Unsere Messungen ergaben einen Free-Rider Anteil (0 Dateien) von 12,48%. 23,36% aller Benutzer bieten 10 oder weniger Dateien an. Genauso sieht es auf der anderen Seite aus: 29,47% aller Benutzer bieten 90% aller Dateien an und 22.88% aller Benutzer 90% des Speichervolumens des Gnutella-Netzes.

Andere Tauschbörsen wie eDonkey oder das weiter unten besprochene Fasttrack-Netz haben geringere Freerider Probleme, weil diese Programme automatisch Daten, die gerade heruntergeladen werden, zum Download frei geben. Erleichtert wird dies dadurch, daß diese Protokolle den Download einer Datei in mehreren Stücken und von mehreren Clients gleichzeitig auf Protokollebene unterstützen.[2]

Einen interessanten Ansatz, das Free-Rider Problem zu bekämpfen, verfolgt - bislang allerdings nur mit wenigen Nutzern - die Tauschbörse Mojonation (http://sourceforge.net/projects/mojo-

1. Es folgt eine Auswertung von ca. 500.000 PONG-Nachrichten, die in einem Zeitraum von drei Tagen an 2 Limewire Ultrapeers aufgezeichnet wurden.
2. Erst neuere Versionen mancher Gnutella-Clients (z.B. Limewire) unterstützen den Download von mehreren Quellen gleichzeitig.

nation/), bei der es eine virtuelle Währung (Mojos) gibt, die man für das Anbieten von Daten und Suchantworten bekommt und zum Suchen und Herunterladen ausgeben kann. Details finden sich in [Aut00;WO02], alternative Anreizsysteme beschreiben [WHS92; CGM02; GLB01]. Während quasi alle Tauschbörsen mit Free-Ridern zu kämpfen haben, so hat das oben geschilderte Gnutella-Protokoll aufgrund seiner dezentralen Struktur mit einem weiteren Problem zu kämpfen: Gnutellas Performance sinkt bei steigender Benutzerzahl. Es gibt verschiedene Arbeiten über die Skalierbarkeit von Gnutella [Rit01; Hon01; Sri01], die im Detail allerdings umstritten sind [Kab01]. Dennoch lässt sich nicht leugnen, dass an der Kritik etwas dran ist. Die Tatsache, dass mit dem oben beschriebenen Gnutella-Protokoll keine Unterscheidung zwischen Knoten mit hoher und niedriger Bandbreite gemacht wurde, führte zu folgenden Problemen, als die Zahl der Gnutella Benutzer Mitte 2000 rapide anstieg, weil bei Napster die ersten Filter installiert wurden:

Mit der Anzahl der Benutzer steigt auch der Bandbreitenbedarf für die Weiterleitung der Gnutella-Protokollnachrichten. Modembenutzer, die mit ca. 40 KBit/sec ans Internet angebunden sind, stellen hier einen besonderen Engpass dar. Sind sie nur mit 4 weiteren Knoten verbunden, so stehen für jede der 4 Verbindungen nur etwa 1 KByte/sec zur Verfügung. Jede eintreffende PING- und QUERY-Nachricht muss nach dem oben beschriebenen Broadcast-Mechanismus an 3 weitere Verbindungen weitergeleitet werden. Es ist schnell einsichtig, dass - selbst ohne Up- oder Downloads oder paralleles Websurfen - die Bandbreite eines Modem/ISDN Benutzers schnell ausgenutzt ist und sich Nachrichten stauen. Dies führte zum vollständigen Zusammenbruch des Gnutella-Netzes im August 2000 [Kil01], als viele Modemknoten im Zentrum des Netzes die Weiterleitung der Nachrichten blockierten. Als Reaktion darauf verhalten sich moderne Gnutella-Clients intelligenter und brechen Verbindungen zu Knoten ab, die überlastet sind. Diese Maßnahme führt dazu, dass Modemknoten an den Rand des Netzes wandern, somit nur relativ wenige Verbindungen unterhalten und keinen Engpass mehr darstellen. Außerdem wurden die Standardeinstellungen für die TTL und die Anzahl der Verbindungen optimiert. Die Anzahl der erreichbaren Benutzer wächst geometrisch mit der Anzahl der Verbindungen und TTL, während Suchnachrichten bei 5 Verbindungen und einer TTL von 5 Hops 1.705 Knoten erreichen, sind es bei 7 Verbindungen und 7 Hops schon 391.909 [Sri01]. Weitere sinnvolle Verbesserungen sind z.B., dass jeder Knoten nur mit einer bestimmten Wahrscheinlichkeit auf einen PING reagiert, dass Knoten PONG-Nachrichten cachen und mit diesen bei einer eintreffenden PING-Nachricht antworten, anstatt die PING-Nachricht weiterzuleiten. Ebenfalls sinnvoll ist es, PING-Nachrichten zu multiplexen, d.h. bei zwei unterschiedlichen kurz

hintereinander eintreffenden PING-Nachrichten die zweite zu verwerfen und stattdessen die PONG Antworten auf die erste auch an den Sender der zweiten zu schicken. Details finden sich in [RF01]. Vielversprechender als diese Maßnahmen ist allerdings die Einführung von Reflektoren und “Super-Peers” bzw. ”Ultra-Peers”, das sind Knoten, die ähnlich einem eDonkey-Server agieren und Dateilisten von Gnutella-Clients empfangen und dann anstelle dieser Clients auf Suchanfragen antworten. Dies reduziert den Weiterleitungsaufwand für QUERY und QUERY-HIT-Nachrichten. Weitere Ideen finden sich in [Oso01]. Eine Übersicht über die Entwicklungsgeschichte von Gnutella gibt [Kil01]. Interessante Messungen im Gnutella-Netz finden sich in [Rip01; RF02; SGG02; Mar02].

Wir haben den Protokolloverhead für einen modernen Gnutella-Client (Limewire 2.6) im Oktober 2002 gemessen. Durch die Verwendung von Ultra-Peers und anderen Maßnahmen ist der Overhead mit 0.1 KByte/sec Verkehr vernachlässigbar gering (ohne diese Maßnahmen liegt der Verkehr mit 23 bis 90 KByte/sec Größenordnungen darüber[1]). Für als Ultra-Peers agierende Knoten ist der Verkehr deutlich größer (40-80 KByte/sec), bei der schnellen Internet Verbindungen dieser Knoten ist dies aber akzeptabel. Durch die geschilderten Verbesserungen ist Gnutella damit wieder konkurrenzfähig mit anderen Protokollen.

5 Kazaa und Grokster – mit dem Fasttrack Protokoll auf der Überholspur

Das momentan erfolgreichste P2P Protokoll ist das von der Firma Fasttrack (http://www.fasttrack.nu/) entwickelte P2P Protokoll, welches von den Clients Kazaa (http://www.kazaa.com/), Kazaa Lite (http://www.kazaalite.tk/) und Grokster (http://www.grokster.com/) verwendet wird, die sich alle zu einem einheitlichen Netzwerk verbinden und gegenseitig Dateien austauschen können.

Tabelle 1 zeigt die Ergebnisse unserer Messungen in der 2. Oktoberwoche 2002: Das Fasttrack Netzwerk führt mit großem Abstand, was die Anzahl der gleichzeitig aktiven Benutzer, die An-

1. Gemessen mit der Gnutella Version von Januar 2001 und 4 bis 12 Verbindungen.

zahl der von ihnen angebotenen Dateien und deren Speicherbedarf betrifft sowie die Anzahl der wöchentlichen Downloads des Clients bei Download.com.

System	# Benutzer	# Dateien	Terabyte	# Downloads
Fasttrack	2,6 Mio.	472 Mio.	3550	4 Mio.
eDonkey	230.000	13 Mio.	650-2600	600.000
Gnutella	120.000	28 Mio.	105	ca. 525.000

Tabelle 1 Messungen der Benutzerzahlen verschiedener Tauschbörsen (10’2002)

Die Fasttrack Zahlen stellen den Wochendurchschnitt über die von Fasttrack selbst ermittelten Zahlen dar. Diese Zahlen dürften tendenziell eher überschätzt sein. Die anderen Zahlen basieren auf Messungen und dürften eher unterschätzt sein. Die eDonkey-Zahlen entstanden durch die Mittelung über die stündliche Auswertung sämtlicher (ca. 250) eDonkey-Server aus der Serverliste unserer eigenen Implementierung. Die Zahlen liegen eng an den Schätzungen von OCB Maurice unter http://ocbmaurice.dyndns.org/pl/ed2k_stats.pl. Die Schätzung der Dateigrößen ist auf diesem Wege mit großer Unsicherheit behaftet. Die Messungen entsprechen allerdings der Erfahrung, dass mit eDonkey häufiger als mit den beiden anderen Tauschbörsen große Dateien getauscht werden. Die Schätzung des gesamten Gnutella-Netzes ist wegen der dezentralen Struktur nicht exakt möglich, die für Gnutella präsentierten Zahlen über die Anzahl der Hosts stimmen mit Angaben der Firma Limewire (http://www.limewire.com/current_size.html) überein. Unsere Messungen basieren auf einer Auswertung an verschiedenen Punkten des Netzes gemessener PONG-Nachrichten zu einem Stichtag (13.10.2002).

Das Fasttrack Protokoll ähnelt in technischer Hinsicht und in der Funktionsweise stark dem eDonkey-Protokoll mit einem aus Benutzersicht allerdings wichtigen Unterschied. Es wird kein Unterschied zwischen einer Server und Clientsoftware gemacht. Das bedeutet nicht, dass das Fasttrack-Netzwerk ohne Knoten mit Serverfunktionalität auskommt. Die Serverfunktionalität ist in die Clientsoftware integriert, womit potenziell jeder Client auch gleichzeitig als Server agieren kann. In der Praxis wird die Serverfunktionalität allerdings nur bei Clients mit einer guten Internetanbindung nach für den Benutzer intransparenten Regeln aktiviert - in der Fasttracksprache wird die Kombination aus Client und Server “Supernode” genannt.
Ein weiterer Unterschied zum eDonkey-Protokoll ist die Tatsache, dass die gesamte Kommunikation verschlüsselt ist, was das Reverse Engineering erschwert. Fasttrack gibt sich auch große Mühe, das Protokoll proprietär zu halten. So hat Fasttrack mit Version 1.3.3 ihres Protokolls die Verschlüsselung geändert, um den Open Source Client giFT (http://gift.sourceforge.net) auszusperren. Die giFT Entwickler haben es mittlerweile aufgegeben, das Fasttrack Protokoll zu knacken und entwickeln stattdessen ihr eigenes OpenFT genanntes Protokoll.

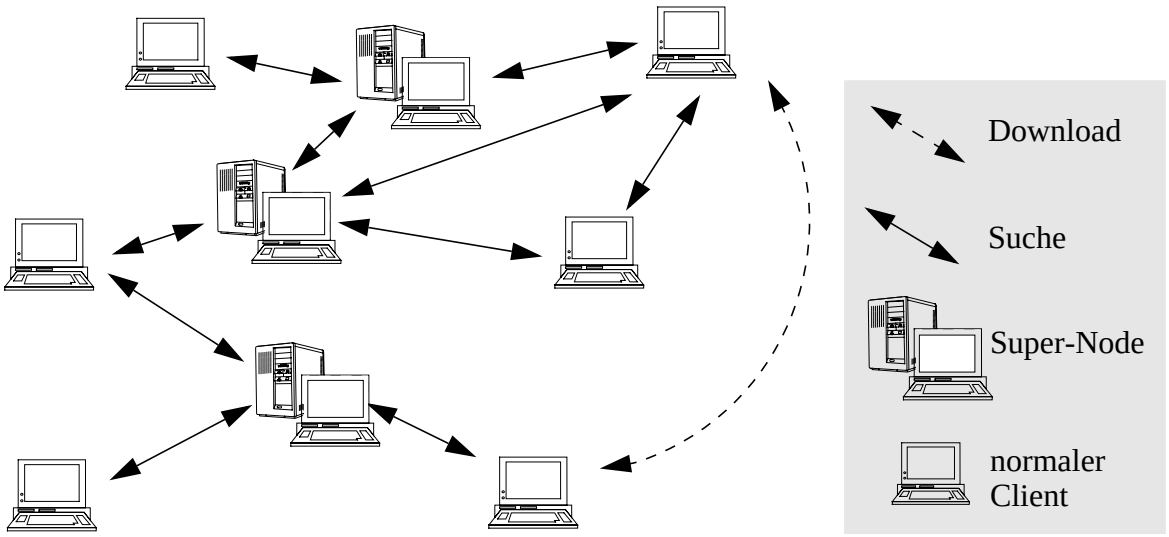


Bild 5 Aufbau des Fasttrack-Netzwerkes

6 Es gibt auch andere Mittel und Wege ...

Über die genannten Tauschbörsen werden zum überwiegenden Teil Dateien ausgetauscht, die urheberrechtlich problematisch sind. Ob vor diesem Hintergrund eine Anonymität der Benutzer wünschenswert sein kann, ist ein strittiges Thema. Unstrittig ist jedoch, dass die meisten Benutzer eine Tauschbörse bevorzugen würden, die ihren Benutzern Anonymität garantiert.

Jeder Rechner im Internet - sofern er nicht durch eine Firewall oder einem Rechner mit Netzwerkadressübersetzung (NAT) geschützt wird - ist durch seine IP-Adresse eindeutig identifiziert. Bei temporären Internetanschlüssen (Modem, ISDN, DSL) wird diese Adresse allerdings üblicherweise nur für die Zeitdauer der Einwahl zugeteilt und danach an andere Benutzer vergeben. Die meisten Provider speichern allerdings ihre Verbindungsdaten über einen längeren Zeitraum (z.B. 30 Tage) ab und können die IP-Adresse eindeutig Benutzern zuordnen (die Datenschutzproblematik lassen wir hier außen vor, diese wird leider in Praxis nur allzu häufig ebenfalls ignoriert). Wenn man daher die IP-Adresse eines Benutzers kennt, der illegale Dateien anbietet, ist es mit Hilfe des Providers (theoretisch) möglich, seinen wahren Namen herauszufinden und ihn zu verklagen. Alle bislang genannten Protokolle schützen die IP-Adressen der beteiligten Personen nur mangelhaft und sind nicht anonym. Spätestens wenn er eine Datei herunterladen will, erfährt der herunterladende Client vom Napster-Server, von den eDonkey-Servern oder anderen Gnutella- oder Fasttrack-Knoten die IP-Adresse und den Port, unter der der Gegenüber die gewünschte Datei anbietet. Dies ist auch nötig, da sonst keine direkte Kontaktaufnahme von Peer zu Peer möglich wäre. Daher garantiert bislang keine der genannten Tauschbörsen ernsthaft die Anonymität ihrer Benutzer.

Es gibt zwar auch P2P Protokolle, bei denen der Datenaustausch verschlüsselt stattfindet (Fasttrack, Aimster, Filetopia). Dieser Mechanismus verhindert, dass Angreifer oder bei Providern installierte Filtersysteme, die den P2P Verkehr abhören, die Suchantworten mit den IP Adresslisten verstehen können. Aber der Mechanismus kann nicht verhindern, dass ein Angreifer, der sich als normaler P2P Client tarnt und normal nach illegalen Inhalten sucht, an diese Listen gerät.

Es gibt zwei Ansätze, mit denen diese Problematik bekämpft wird. Der erste Ansatz ist z.B. in der Tauschbörse Filetopia (http://www.filetopia.org/) implementiert. Dort besteht die Möglichkeit, für einen Dateitransfer einen Proxy, also eine Art Zwischenhändler, zu verwenden (Bouncer genannt). Über diesen Proxy wickeln eine Vielzahl von Benutzern ihre Transfers ab. Lediglich der Proxy kennt die beiden IP-Adressen der an einem Download beteiligten Clients während jeder Client nur die des Proxies erfährt. Der Dateiaustausch findet verschlüsselt statt und nur die beiden Clients kennen den Schlüssel, womit der Betreiber des Proxies wiederum abstreiten kann, sich der Illegalität der Datei bewusst gewesen zu sein. Damit dieses Prinzip funktioniert, muss es allerdings eine Anzahl leistungsfähiger Proxies geben und genau diese waren zum Zeitpunkt unserer Messungen (Oktober 2002) noch sehr dünn gesät.

Einen anderen Ansatz verfolgt Freenet (http://freenetproject.org). Freenet ist eigentlich weniger als Tauschbörse für kopiergeschützte Inhalte gedacht gewesen, denn als robuste verteilte Plattform, um Dokumente anonym anbieten zu können und Zensur zu verhindern. Freilich kann man argumentieren, dass etwas, das Freiheitskämpfern in der dritten Welt gelegen kommt, auch für die Anbieter von Kinderpornographie interessant sein kann; dies hat Freenet einen etwas zweifelhaften Ruf eingetragen. Es darf allerdings bezweifelt werden, dass über Freenet überhaupt größere Datenmengen ausgetauscht werden, da es mit Performanceproblemen und instabiler Software zu kämpfen hat. Das hinter Freenet stehende Konzept ist dennoch einer genaueren Betrachtung wert. Jeder am Freenet beteiligte Knoten speichert Daten zwischen, die von anderen Knoten ins Netz gestellt wurden. Freenet stellt sicher, dass der Betreiber eines Knotens die Urheberschaft der bei ihm gespeicherten Daten und die Kenntnis ihres Inhalts glaubhaft abstreiten kann. Dateien werden über einen eindeutigen Schlüssel identifiziert, die Berechnung dieses Schlüssels ist eine etwas ausgeklügeltere Variante des Hashwertes im eDonkey-Protokoll [CSWH02]. Da Knoten nur den Schlüssel (Hash) und die Daten kennen, können sie keine direkten Rückschlüsse auf den Dateinamen und das Format der Daten ziehen. Freenet hat keinen Suchmechanismus, d.h. man muss auf einem anderem Weg den Schlüssel der gewünschten Datei herausfinden, z.B. über das WWW oder Newsgroups. Soll eine Datei, von der der Schlüssel

bekannt ist, heruntergeladen werden, so reicht der Knoten, der die Datei herunterladen will, den Schlüssel an seinen Nachbarknoten weiter. Dieser leitet die Nachfrage wiederum weiter, sofern er sie nicht selbst befriedigen kann. Erreicht die Nachfrage schließlich einen Knoten, der die Daten anbietet, so schickt dieser sie an den Vorgängerknoten. Dieser speichert die Daten in seinem Cache zwischen und leitet sie wiederum an den Vorgängerknoten weiter, bis sie an den ursprünglich nachfragenden Knoten gelangen. Es ist unmittelbar einsichtig, dass diese Art des Hop-by-Hop Datentransfers deutlich mehr Verkehr verursacht als der direkte Datentransfer zwischen 2 Clients wie bei eDonkey, Gnutella oder Fasttrack. Außerdem ist die Downloadgeschwindigkeit auf die minimale Geschwindigkeit in allen Gliedern dieser Kette beschränkt und die Wahrscheinlichkeit für einen Abbruch des Downloads steigt auch mit der Länge dieser Kette.

Dadurch, dass die Daten auf jedem Hop gecacht werden, wandern sie quasi in die Richtung, aus der sie häufig nachgefragt werden. Außerdem sind so häufig nachgefragte Daten in vielen Caches und daher meist schneller zu erreichen. Da alle Daten in Caches gehalten werden, kann es vorkommen, dass selten nachgefragte Dateien im Laufe der Zeit aus allen Caches gelöscht werden und verschwinden. Im Unterschied zu den anderen vorgestellten Tauschbörsen haben die Betreiber der Knoten also keine Kontrolle über die bei ihnen gespeicherten Daten, Zensur ist damit erschwert.

Daten werden auf ähnlichem Weg in das Netzwerk eingespeist. Zunächst berechnet der einspeisende Knoten den eindeutigen Schlüssel der Datei und schickt eine “Einfügen-Nachfrage” an einen Nachbarknoten. Dieser überprüft, ob der Schlüssel für ihn noch frei ist. Ist dies der Fall, reicht er die Nachfrage wiederum an Nachbarknoten weiter, bis eine vorgegebene Anzahl Hops erreicht ist. Wird keine Schlüssel-Kollision entdeckt, so speist der Autor die Datei in das Netz ein: Die Datei wird bei den Rechnern zwischengespeichert, bei denen auch nach dem Schlüssel nachgefragt wurde. Der Autor kann die Datei jetzt lokal löschen und die Urheberschaft an dem Dokument leugnen. Genauere Informationen finden sich in [Lan01] und [CSWH02]. Forschungsergebnisse zum Thema Anonymität und Zensur in P2P Systemen finden sich in [FS02; HW02; MFSM02]. Während Freenet einen interessanten Ansatz bietet, Zensur zu verhindern, ließe sich der Mechanismus von Filetopia relativ leicht in die oben besprochenen Tauschbörsen eDonkey/Overnet, Gnutella und Fasttrack einbauen.

7 Fazit

P2P-Netze bieten ein spannendes und reichhaltiges Forschungsumfeld. So gibt es beispielsweise einige Forschungsanstrengung über die effiziente Suche in P2P-Netzwerken. Ansätze wie CAN [RFH+01], Chord [SMK+01; DBK+01], Pastry und Tapestry [PRR97; DR01c; ZKJ01] setzen dabei ein stark strukturiertes Netzwerk voraus. [LCC+02] gehen auf die Suche in unstrukturierten P2P-Netzen ein. Weitere Arbeiten in diesem Bereich sind [FV02; IRF01, YGM02; RV01; SGM02].
Terradir [BKS01] und Oceanstore [Wel00; BCE+99] sowie die weiteren Arbeiten [DKK+01; DR01a; DR01b] beschäftigen sich mit der Verwendung von P2P-Netzen zur Schaffung von verteilten globalen Dateisystemen.
Die Verbindungen zwischen den P2P Knoten formen ein so genanntes Overlay-Netz über das Internet IP-Netzwerk. Die Topologie dieses Netzes beeinflusst die Performance der P2P-Netzes. Mit den Eigenschaften sinnvoller Topologien (z.B. Small-World Graphen) beschäftigen sich unter anderem [PU01; KBS02; KBS00; LNS01; BKR99]. Weitere interessante Ideen finden sich in [LNB02; GBHC01; MM02; ZZJ+01], eine gute Taxonomie beschreibt [CP02].

Dieser Artikel erläuterte die technischen Hintergründe sowie die Funktionsweise der bekanntesten P2P Tauschbörsen Napster, eDonkey, Gnutella und Fasttrack. Zu den genannten Protokollen wurden verschiedene Messergebnisse dargestellt.
Vergleicht man die Entwicklungsrichtung der Protokolle, so verlaufen diese in die gleiche Richtung. Moderne leistungsfähige P2P Tauschbörsen verwenden die gleichen Prinzipien:

- Dateihashes zur Identifikation von identischen Dateien, was den gleichzeitigen Download von mehreren Quellen und die Fortsetzung unterbrochener Downloads erleichtert.
- Supernodes oder explizite Server zur effizienten Suche und Mechanismen, die die Konnektivität trotz Ausfall mehrer dieser Knoten sicherstellen.

Gezeigt wurde auch, warum bislang - von wenig erfolgreichen Spezialanwendungen wie den ebenfalls vorgestellten Anwendungen Freenet und Filetopia abgesehen - die Anonymität der Tauschbörsenteilnehmer nicht wirklich geschützt ist. Der Schutz der Anonymität wird unserer Meinung nach das hervorstehende Kennzeichen der nächsten Generation von P2P Tauschbörsen sein.

Danksagung

Die Autoren danken Axel Bock, Andreas Hebel und Vasilios Darlagiannis für ihre Mithilfe und Diskussionsbeiträge.

8 Literatur

[AH00] *E. Adar und B. A. Huberman*. Free Riding on Gnutella. First Monday Peer-Reviewed Journal on the Internet, 5(10), Oktober 2000. http://www.firstmonday.dk/issues/issue5_10/adar/index.html.

[Aut00] *Autonomous Zone Industries / Mojo Nation Project.* Technology Overview, Februar 2000. http://www.transarc.ibm.com/ ota/others-papers/MojoNation_TechnicalOverview.html, Abruf am 2002-10-08.

[BCE+99] *D. Bindel, Y. Chen, P. Eaton, R. Gummadi D. Geels, S. Rhea, H. Weatherspoon, C. Wells W. Weimer, B. Zhao und J. Kubiatowicz.* Oceanstore: An extremely wide-area storage system. Technical Report UCB/CSD-00-1102, UC Berkeley, Mai 1999.

[BKR99] *F. Baccelli, D. Kofmanund J. Rougier.* Self-organizing hierarchical multicast trees and their optimization. In: Proceedings IEEE Infocom'99, New York, USA, 1999.

[BKS01] *B. Bhattacharjee, P. Keleher und B. Silaghi.* The design of terradir. Technical Report CS-TR-4299, University of Maryland, Oktober 2001.

[CGM02] *B. F. Cooper und H. Garcia-Molina.* Peer-to-peer resource trading in a reliable distributed system. In: Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02), Cambridge, MA, USA, S. 319–327. Springer LNCS 2429, März 2002.

[Cli00] *Clip2 Project.* The Gnutella Protocol Specification v0.4 - Document Revision 1.2, März 2000. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf, Abruf am 2002-10-08.

[CP02] *J. Crowcroft und I. Pratt.* Peer to Peer: Peering into the Future. In: Proceedings of the IFIP-TC6 Networks 2002 Conference, Pisa, Italy. Springer LNCS 2345, Mai 2002.

[CSWH02] *I. Clarke, O. Sandberg, B. Wiley und T. W. Hong.* Freenet: A Distributed Anonymous Information Storage and Retrieval System. In: Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability. Springer LNCS 2009, 2002.

[DBK+01] *F. Dabek, E. Brunskill, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica und H. Balakrishnan.* Building peer-to-peer systems with chord, a distributed lookup service. In: Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII), Mai 2001.

[DKK+01] *F. Dabek, M. F. Kaashoek, D. Karger, R. Morris und I. Stoica.* Wide-area cooperative storage with CFS. In: Proceedings of the 18th ACM Symposium on Operating Systems Principles(SOSP'01), Chateau Lake Louise, Banff, Canada, Oktober 2001.

[DR01a] *P. Druschel und A. Rowstron.* PAST: A large-scale, persistent peer-to-peer storage utility. In: Proceedings of Eighth ACM Workshop on Hot Topics in Operating Systems (HotOS-VIII), Schloss Elmau, Germany, Mai 2001.

[DR01b] *P. Druschel und A. Rowstron.* PAST: Persistent and anonymous storage in a peer-to-peer networking environment. In: Proceedings of Eighth ACM Workshop on Hot Topics in Operating Systems (HotOS-VIII), Schloss Elmau, Germany, Mai 2001.

[DR01c] *P. Druschel und A. Rowstron.* Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, S. 329–350, 2001.

[Dump] *TCPDump Open Source Project.* TCPDump Software. Netzwerk Verkehrsanalyse Software, http://www.tcpdump.org/, Abruf am 2002-10-08.

[Elson] *J. Elson*. TCPFlow Software. Netzwerk Verkehrsanalyse Tool, http://www.circlemud.org/jelson/software/tcpflow/, Abruf am 2002-10-08.

[FGM+99] *R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L Masinter, P. Leach und T. Berners-Lee.* Hypertext Transfer Protocol - HTTP/1.1. Draft Standard RFC 2616, Juni 1999.

[FS02] *A. Fiat und J. Saia.* A Price Communication Protocol. In: Proceddings of Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, Januar 2002.

[FV02] *M. J. Freedman und R. Vingralek.* Efficient peer-to-peer lookup based on a distributed trie. In: Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02), Cambridge, MA, USA, S. 66–75. Springer LNCS 2429, März 2002.

[GBHC01] *S. D. Gribble, E. A. Brewer, J. M. Hellerstein und D. Culler.* Scalable, distributed data structures for internet service construction. In: Proceedings of the Fourth USENIX Symposium on Operating Systems Design and Implementation (OSDI 2000), September 2001.

[GLB01] *I. M. Philippe Golle und K. Leyton-Brown.* Incentives for sharing in peer-to-peer network. In: Proceedings of WELCOM'01, 2001.

[Hon01] *T. Hong.* Performance. In: Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology (Editor A. Oram), S. 203–241. O'Reilly & Associates, März 2001.

[HW02] *S. Hazel und B. Wiley.* Achord: A variant of the chord lookup service for use in censorship resistant peer-to-peer publishing systems. In: Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02), Cambridge, MA, USA. Springer LNCS 2429, März 2002.

[IRF01] *A. Iamnitchi, M. Ripeanu und I. Foster.* Locating data in (small-world?) peer-to-peer scientific collaborations. In: Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02), Cambridge, MA, USA), März 2001.

[Kab01] *M. Kabanov.* In Defense of Gnutella, February 2001. Online-Artikel, http://www.gnutellameter.com/gnutella-editor.html, Abruf am 2002-10-08.

[KBS00] *P. Keleher, B. Bhattacharjee und B. Silaghi.* On network-aware clustering of web clients. In: ACM SIGCOMM 2000, S. 97–110, 2000.

[KBS02] *P. Keleher, B. Bhattacharjee und B. Silagh.* Are virtualized overlay networks too much of a good thing? In: Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02), Cambridge, MA, USA, März 2002.

[Kil01] *F. Kileng.* Peer-to-Peer file sharing technologies - Napster, Gnutella and beyond. Technical Report R&D Report 18/2001, Telenor, Juli 2001. http://www.telenor.no/fou/publisering/Rapp01/R18_2001.PDF, Abruf am 2002-10-08.

[Lan01] *A. Langley.* Freenet. In: Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology (Editor A. Oram), S. 94–122. O'Reilly & Associates, März 2001.

[LCC+02] *Q. Lv, P. Cao, E. Cohen, K. Li und S. Shenker.* Search and replication in unstructured peer-to-peer networks. In: Proceedings of SIGMETRICS 2002, S. 258–259, 2002.

[Lim01] *Lime Wire LLC.* Limewire: Network Improvements, März 2001. Technical Report, http://www.limewire.com/index.jsp/net_improvements, Abruf am 2002-10-08.

[LNB02] *D. K. D. Liben-Nowell und H. Balakrishnan.* Observations on the dynamic evolution of peer-to-peer networks. In: Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02), Cambridge, MA, USA, März 2002.

[LNS01] *J. Liebeherr, M. Nahas und W. Si.* Application-layer multicast with delaunay triangulation overlays. Technical Report CS-2001-28, University of Virginia, Department of Computer Science, November 2001.

[Mar02] *E. P. Markatos.* Tracing a large-scale peer to peer system: an hour in the life of gnutella. In: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2002.

[MFSM02] *J. C. Michael, J. Freedman, Emil Sit und R. Morris.* Tarzan: A peer-to-peer anonymizing network layer. In: Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02), Cambridge, MA, USA. Springer LNCS 2429, März 2002.

[MM02] *P. Maymounkov und D. Mazieres.* A peer-to-peer information system based on the xor metric. In: Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02), Cambridge, MA, USA, März 2002.

[Ope00] *Open Napster Project.* Napster Messages, April 2000. http://opennap.sourceforge.net/ napster.txt, Abruf am 2002-10-08.

[Oso01] *S. Osokine.* The Flow Control Algorithm for the Distributed 'Broadcast-Route' Networks with Reliable Transport Links, Januar 2001. Online-Artikel, http://www.grouter.net/gnutella/flowcntl.htm, Abruf am 2002-10-08.

[PRR97] *C. G. Plaxton, R. Rajaraman und A. W. Richa.* Accessing nearby copies of replicated objects in a distributed environment. In: Proceedings of ACM Symposium on Parallel Algorithms and Architectures, S. 311–320, 1997.

[PU01] *P. R. G. Pandurangan und E. Upfal.* Building low-diameter p2p networks. In: Proceedings of the 42nd annual IEEE Symposium on the Foundations of Computer Science (FOCS), 2001.

[RF01] *C. Rohrs und V. Falco.* Limewire: Ping Pong Scheme, April 2001. Technical Report, http:// www.limewire.com/index.jsp/pingpong, Abruf am 2002-10-08.

[RF02] *M. Ripeanu und I. Foster.* Mapping the gnutella network: Macroscopic properties of large-scale peer-to-peer systems. In: Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02), Cambridge, MA, USA, S. 85–93. Springer LNCS 2429, März 2002.

[RFH+01] *S. Ratnasamy, P. Francis, M. Handley, R. Karp und S. Shenker.* A Scalable Content-Addressable Network. In: Proceedings of the ACM SIGCOMM'01 Conference, San Diego, California, 2001.

[Rip01] *M. Ripeanu.* Peer-to-Peer Architecture Case Study: Gnutella Network. In: Proceedings of IEEE 1st International Conference on Peer-to-peer Computing (P2P2001), Linkoping, Sweden. IEEE, August 2001.

[Rit01] *J. Ritter.* Why Gnutella Can't Scale - No, Really, February 2001. Online-Artikel, http:// www.darkridge.com/ jpr5/doc/gnutella.html, Abruf am 2002-10-08.

[Riv90] *R. Rivest.* The MD4 Message Digest Algorithm. RFC 1186, Oktober 1990.

[RV01] *P. Reynolds und A. Vahdat.* Efficient peer-to-peer keyword searching. Technical Report 2002-15, Duke University, September 2001.

[SGG02] *S. Saroiu, P. Gummadi und S. Gribble.* Measurement study of peer-to-peer file sharing systems. In: Proceedings of Multimedia Computing and Networking 2002 (MMCN'02), San Jose, CA, USA. SPIE, Januar 2002.

[SGM02] *Q. Sun und H. Garcia-Molina.* Partial lookup services (extended version. Technical Report 2002-15, Stanford University, Februar 2002.

[SMK+01] *I. Stoica, R. Morris, D. Karger, M. Kaashoek und H. Balakrishnan.* A Price Communication Protocol. In: Proceedings of the ACM SIGCOMM'01 Conference, San Diego, California, 2001.

[Sri01] *K. Sripanidkulchai.* The popularity of Gnutella queries and its implications on scalability, Februar 2001. Diskussionspapier, http://www.cs.cmu.edu/ kunwadee/research/p2p/gnutella.html, Abruf am 2002-10-08.

[SW02] *S. Sen und J. Wang.* Analyzing Peer-to-Peer Traffic Across Large Networks. In: Proceedings of ACM SIGCOMM Internet Measurement Workshop, Marseille, France, November 2002. http:// www.icir.org/vern/imw-2002/imw2002-papers/167.ps.gz, Abruf am 2002-10-08.

[Wel00] *C. Wells.* The oceanstore archive: Goals, structures and self-repair. Master Thesis, Mai 2000.

[WHS92] *C. A. Waldspurger, T. Hogg und W. Stornetta.* Spawn: a distributed computational economy. IEEE Transactions Software Engineering, 18(2), S. 103–117, 1992.

[WO02] *B. Wilcox-O'Hearn.* Experiences Deploying a Large-Scale Emergent Network. In: Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02), Cambridge, MA, USA, S. 104–110. Springer LNCS 2429, März 2002. http://www.cs.rice.edu/Conferences/IPTPS02/188.pdf, Abruf am 2002-10-08.

[YGM02] *B. Yang und H. Garcia-Molina.* Efficient Search in Peer-to-peer Network. In: Proceedings of ICDCS 2002, 2002.

[ZKJ01] *B. Y. Zhao, J. D. Kubiatowicz und A. D. Joseph.* Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

[ZZJ+01] *S. Zhuang, B. Zhao, A. Joseph, R. Katz und J. Kubiatowicz.* Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In: Proceedings of the eleventh International Workshop on Network and Operating Systems Support for Audio and Video (NOSSDAV'01), Port Jefferson, New York, 2001.