Filesharing-Systeme

Universität Leipzig

Problemseminar peer-to-peer data management
2003 Wintersemester

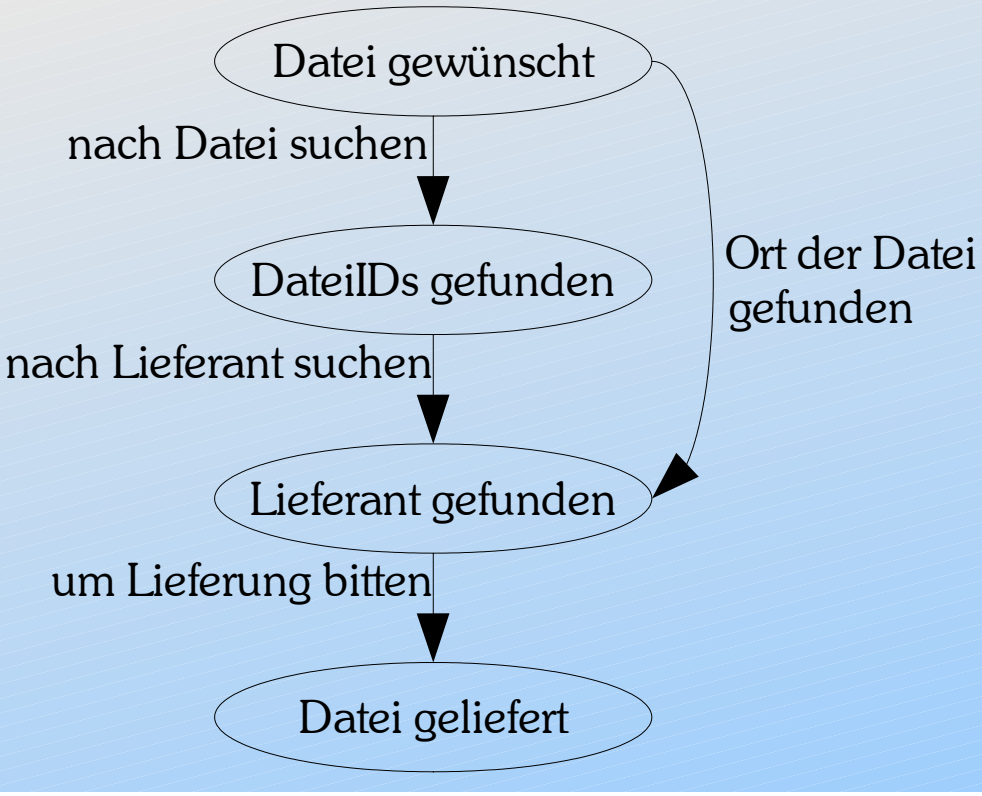
Xuân Baldauf
<xuan--peer2peer-data-management@studium.baldauf.org>

Themen des Vortrags

- Funktionsprinzip
- Napster
- Gnutella
- eMule
- BitTorrent
- Ausblick

Das Skript enthält mehr Themen als der Vortrag

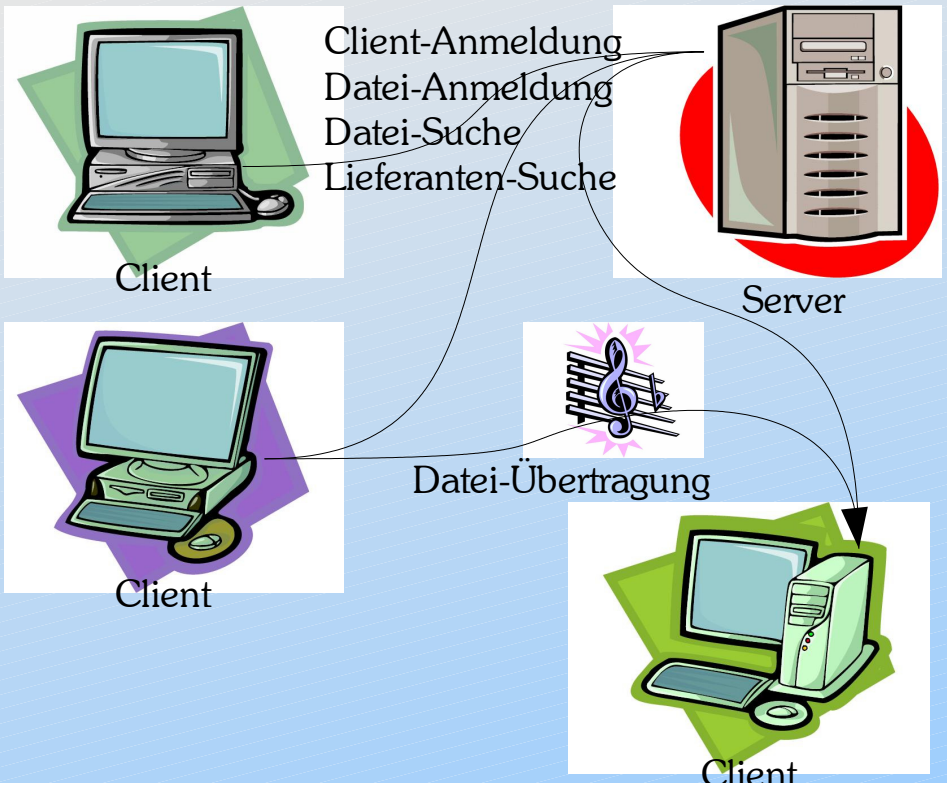
Funktionsprinzip (aus Nutzersicht)



Filesharing Live

- für jedes Netzwerk:
  - mit Netzwerk verbinden
  - Datei suchen
  - Download starten

Napster



Gnutella

- Idee: Filesharing ohne zentrale Instanz muss möglich sein

➔ Jeder Teilnehmer muss alle Rollen übernehmen können

  - Such-Anfrage beantworten
  - Datei senden (Lieferant)
  - Datei empfangen (Bezieher)
  - Netzwerk-Teilnehmer finden
  - Routing

Gnutella: Architektur

- 4 Nachrichten-Typen:
  - *Ping*: „Hallo, ist da wer? Hier ist Otto.“
  - *Pong*:
    - „Hallo Otto, hier antwortet das Echo. Mein Ort ist … .“
  - *Query*:
    - „Hat jemand eine Datei mit „Linux“ im Namen?“
  - *QueryHits*:
    - „Ich (mein Ort ist …) habe folgende Dateien:
      - meine Datei Nummer 578 ist „SuSE Linux 9 CD 1.iso“, Größe 649.85MB
      - meine Datei Nummer 8919 ist „Mandrake Linux 9.2 CD2 i586.iso“, Größe …
      - …

Gnutella: Routing

- Fluten („flooding“)
  - Weiterleitung an alle Peers
  - (also kein wirkliches Routing)
- jedes Paket hat
  - PaketID
  - AbsenderID
  - time-to-live (wird von jedem Hop dekrementiert)
  - hop count (wird von jedem Hop inkrementiert)
- Standard time-to-live: 7

Gnutella: Teilnehmer finden

- Algorithmus:
  - Es gebe eine nichtleere Liste von Teilnehmern.
  - Man verbinde sich zu einem dieser Teilnehmer.
  - Man sende ein *Ping*-Paket aus.
  - Man warte auf die *Pong*-Antworten.
  - Man nehme alle Absender von Pong-Antworten in die Liste der Teilnehmer auf.
- Teilnehmerliste füllt sich
- Teilnehmerliste bleibt aktuell

Problemseminar peer-to-peer data management: Filesharing-Systeme
09. Dezember 2003

Gnutella: Im Netzwerk bleiben

- Algorithmus: (simpel)
  - Sobald mit weniger als 4 Teilnehmern verbunden:
    - Verbinde mit neuem Teilnehmer aus Teilnehmerliste
- Wenn neu verbunden, dann wahrscheinlich mit beschäftigtem Teilnehmer
- Beschäftigte Teilnehmer beenden die Verbindung.
- Unbeschäftigte Teilnehmer lassen Verbindung bestehen.

→ Ort im Gnutella-Netzwerk stabilisert sich.

Gnutella: Datei-Suche

- Aussenden eines *Query*-Pakets
  - enthält Schlüsselwörter, die zur gesuchten Datei passen
- Wenn Upload-Kapazität bei einem Lieferanten frei:
  - Aussenden eines *QueryHits*-Pakets an den Fragenden
  - enthält Socket-Adresse des Lieferanten
  - enthält eine Liste von möglichen passenden Dateien
    - für jede Datei
      - Datei-Name, -Größe, ...
      - Nummer der Datei beim Lieferanten

Gnutella: Datei-Transfer

- Aufbau einer direkten HTTP-Verbindung zum Lieferanten:
  - GET /578/SuSE%20Linux%209%20CD%201.iso/ HTTP/1.1
    User-Agent: einGnutellaTeilnehmer/0.1
    Range: bytes=123456789-
    ...
  - HTTP/1.1 200 OK
    Server: nochEinGnutellaTeilnehmer/0.1
    Content-Type: application/binary
    Content-Length: 557957867
    ...

Gnutella: Bootstrapping

- Wie findet man einen lebenden Gnutella-Teilnehmer?
  - Öffentlich bekannter Teilnehmer mit fester Adresse:
    - Sendet nur *Pong*-Pakete und schließt danach die Verbindung.
  - gespeicherte Teilnehmer-Liste
    - ist aber nach Monaten veraltet
  - GWebCache
    - Enthält eine Liste von aktuellen Teilnehmern
    - Wird von Teilnehmern selbst automatisch aktualisiert
    - GWebCache-Instanzen sind per Web-Suchmaschine lokalisierbar

Gnutella: Bewertung

- schlecht:
  - Hop-Grenze erzeugt individuellen Horizont
  - Paket-Überschwemmung verbraucht Bandbreite
  - Netzlast wächst quadratisch mit Teilnehmer-Zahl
- gut:
  - Passive Suche möglich
    - Auswertung der weitergeleiteten *QueryHits*-Pakete
  - sehr hohe Ausfallsicherheit

Gnutella: Schlussfolgerung

- „Proof-of-concept“ ist erreicht: dezentrales Filesharing funktioniert.
- Gnutella ist kaum verwüstlich
- Viele ungelöste Probleme, die Effizienz verhindern
- viele Features (Authentifizierung, DateiID, ...) fehlen
- offenes Protokoll mit freien Implementationen
  - wird gerne von der Wissenschaft untersucht
- Lösungsansätze zu Problemen sind in Entwicklung

eMule

- ist heutzutage sehr populär
- eMule-Netzwerk besteht aus Teil-Netzen
  - jedes Teilnetz hat einen Server
  - und viele Clients
- eMule-Teilnetze sind strukturell ähnlich zu Napster
- aber eMule bietet viele nützliche Features und Konzepte

eMule: Architektur

- Clients senden Anfragen an Server
- Server sendet diese Anfragen nicht weiter
  - Ergebnisse sind damit
    - unvollständig
    - aber schnell verfügbar
- Server versorgt Client mit aktueller Server-Liste
- von Zeit zu Zeit: Verbindungs-Trennung
  - so Replikation über Teilnetz-Grenzen hinweg

eMule: DateiID

- Für jede Datei: eindeutige Nummer
  - Implementation: MD4-Hash-Wert als DateiID
  - ist nur pseudo-eindeutig, aber trotzdem recht brauchbar
- Suche verläuft mehrstufig:
  - Suchanfrage → DateiIDs
  - DateiID → Lieferanten

eMule: DateiID: Vorteile

- Trennung zwischen Such-Zeit und Download-Zeit
- Lieferanten-Liste veraltet mit der Zeit
  - mit DateiID kann Lieferanten-Liste erneuert werden

→ Ermöglichung von

- langwierigen Downloads (große Dateien)
- Download seltener Dateien (Lieferant oft nicht erreichbar)
- Speicherung einer „Wunschliste“ über längere Zeit

→ Große Erhöhung des Nutzens

eMule: kooperativer Download

- Unterteilung von Dateien in Segmente
- Segment-weiser Download
  - Wahl der Segmente: zufällig
  - → unterschiedliche Segmente bei unterschiedlichen Clients
- fertige Segmente werden automatisch freigegeben

→ Clients laden fehlende Segmente von sich gegenseitig

- Original-Lieferant wird entlastet

eMule: Warteschlange

- Wenn zu hohe Nachfrage
  - Gnutella: gar kein Angebot (keine *QueryHits*-Antwort)
  - Napster: Ablehnung der Download-Bitte
  - eMule: anfügen an Warteschlange
- Vorteil:
  - prinzipielle Existenz von Dateien wird nicht verdeckt
  - Stürme von Download-Bitten werden vermieden
- Prinzip: first come → first serve

eMule: gegenseitige Credits

- Bezieher A merkt sich Menge an heruntergeladenen Daten (für jeden Client)
  - Gutschrift für den Client B
- Ist B in der Warteschlange von A, so wandert B schneller vorwärts als normal, abhängig vom Credit.
- kooperative Downloader geben sich gegenseitig Credit

→ *Leecher* müssen länger warten

→ Kooperation erhöht eigenen Nutzen

- Credit ist nicht auf eine Datei beschränkt

eMule: Lieferanten-Listen-Austausch

- Auf Download Anfrage durch Client A an Client B: B liefert Liste aller ihm bekannten Lieferanten
- Vervollständigung der Lieferanten-Liste bei A
- funktioniert über Server-Grenzen hinweg
  - da (von Zeit zu Zeit) Wechsel des Servers (bei Beibehaltung der Lieferanten-Liste)
- macht unabhängiger von Server und Server-Kapazität

eMule: Dateinamen-Austausch

- Bezieher erhält Dateinamen von jedem Lieferanten
- Dateinamen unterscheiden sich
- Variation der Dateinamen normalerweise gering
- Variation der Dateinamen groß bei *Fakes*
  - da oft Einsatz Fakes für verschiedene Originale
- mehrere Namensvorschläge
  - → geben besseren Aufschluss über vermutlichen Inhalt

eMule: (anti)soziales Verhalten

- hohe Download-Bandbreite bei niedriger Upload-Bandbreite
- Suchanfrage an alle Server statt nur an einen
- maskierter Teilnehmer (Firewall), da Koordination durch Server statt direkt
- Bewusste Benachteiligung vermeintlich antisozialer Teilnehmer
  - langsamerer Fortschritt in der Warteliste
  - Ausschluss aus Server oder Warteliste

eMule: Schlussfolgerung

- strukturell nicht viel besser als Napster
  - theoretisch schlechte Ausfallsicherheit
  - theoretisch schlechte Skalierbarkeit
  - Server: keine freie Software
- praktische Skalierbarkeit aber erstaunlich gut: 250'000 Clients gleichzeitig auf einem Server
- Angehen der *Leech*-Problematik
- DateiID: „wesentlicher Meilenstein“

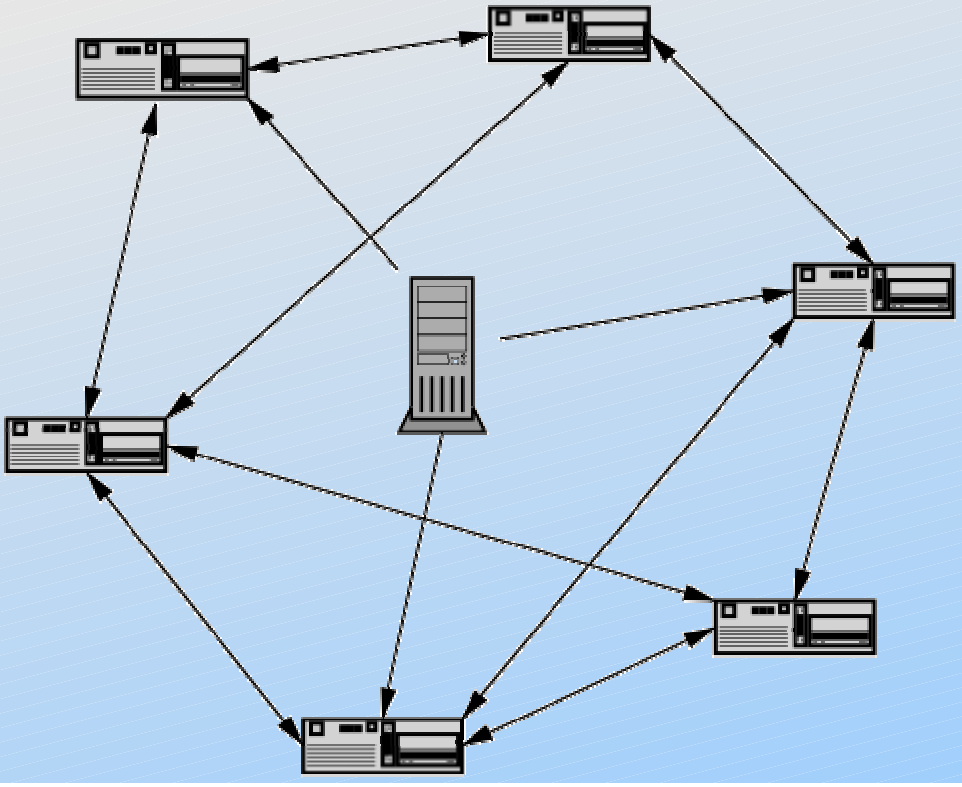
BitTorrent

- Normale Datei-Distribution per FTP oder HTTP:
  - Je höher die Nachfrage, desto niedriger die Download-Geschwindigkeit
- Gedanke
  - Wer downloaded, hat bereits einige Segmente der Datei
  - Wer downloaded, kann diese Segmente an andere Downloader senden
  - Entlastung der ursprünglichen Server möglich

BitTorrent: Architektur

- Für jede Datei ein extra Filesharing-Netzwerk
  - *Tracker*: Zentrum des individuellen Netzwerkes
- *Tracker:* kennt alle Clients
- Clients haben in ihrer Gesamtheit die Datei
- ein Client (*Origin*) hat die Datei von Anfang an
- Clients beliefern sich untereinander.
- Wenn kein anderer Client verfügbar, dann Lieferung nur von *Origin*

BitTorrent: Architektur



BitTorrent: kooperativer Download

- Kooperation ähnlich *e*Mule
- eingehende Lieferung wird **sofort** mit ausgehender Lieferung positiv beantwortet
  - „Wie du mir, so ich dir“
  - entspricht „tit-for-tat“-Strategie aus Spieltheorie
  - ist eine der optimalsten Kooperations-Strategien
- für *Leecher* starkes negatives Feedback (Warten)
- für kooperierende artner hohe Download-Raten
  - über 100KB/s ist normal

BitTorrent: Web-Integration

- Vorgehensweise
  - Klick auf BitTorrent-Link
  - Download einer .torrent-Datei
  - Start des BitTorrent-Clients
- keine Suche nach freigegeben Dateien vorgesehen
  - gute Eignung für Distribution
  - schlechte Eignung für privates Filesharing
- ähnliche Funktionalität auch bei eMule (aber dort nicht so hohe Download-Raten)

Ergebnisse der Live-Demo

- ja, schauen wir mal...

Ausblick

- nur wenige (populäre) Filesharing-Systeme vorgestellt
- viele neue Ansätze werden untersucht und implementiert
- spannende Features:
  - übertragbare Credits
  - verteilte Relationen (globale, verteilte Datenbank)
  - anonymes Publizieren
  - gegenseitiges verschlüsseltes Backup