Nebenläufige und verteilte Programmierung

peer-to-peer

Jörg Naß

Seminar im Wintersemester 2003/04

Einleitung:

Peer-to-peer:

- Internet Applikationen
- große Anzahl von unabhängigen Teilnehmern

- gleichberechtigte, selbst organisierte Netzwerke
- nutzen konventionelle Internet-Protokolle
- keine zentralisierte Struktur

Bereiche:

P2P und Grid Computing

- Ermöglichen der gemeinsamer Nutzung von Ressourcen in verteilten Gemeinschaften

P2P Systeme haben sich auf Ressourcenteilung in Umgebungen konzentriert, mit:

- möglicherweise Millionen von Nutzern
- einheitlichen / ähnlichen Desktopsystemen

  (dementsprechend ähnliche Ressourcen)
- niedrigere Bandbreite

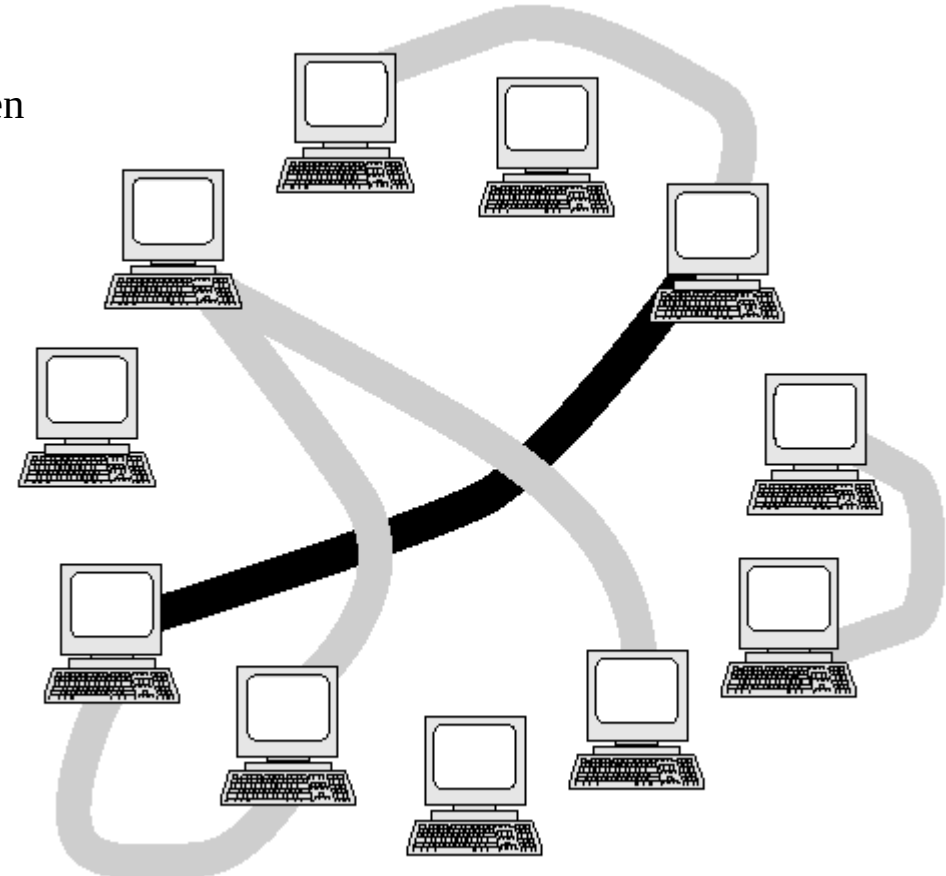
  (meistens per Einwahl)

Grid:

- im allgemeinen kleinere Gruppen
- mit besseren Anbindungen
- unterschiedlicheren Ressourcen

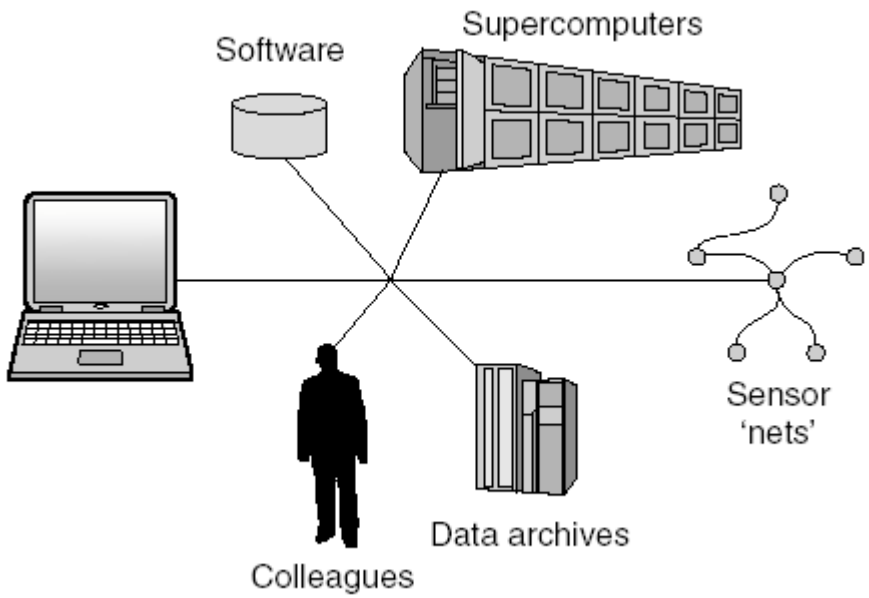
Peer-to-peer

- globale gemeinsame Nutzung von Ressourcen
- Gleiche Ressourcen in einer gegenseitigen misstrauischen Umgebung
- oft für eine spezielle Aufgaben erstellt, z.B.:
  - file-sharing
  - Zahlen Faktorisierung
  - nach außerirdischem Leben suchen (SETI@Home)



Grid

- Flexibler, leistungsstarker, gut verfügbarere Zugang zu allen benötigten Ressourcen
- viele verschiedene und speziellere Ressourcen
- sichere Umgebung
- Erstellung von leistungsfähigen virtuellen EDV-Systemen



P2P:

- Anmerkungen zu P2P
- vom Client-Server Model zu peer-to-peer
- Anwendungsgebiete:
  - Ablage/Speicherung
  - Berechnung
  - Suche

Anmerkungen zu P2P:

- Erste Erwähnung von peer-to-peer vor über 25 Jahren

  in einem Dokument von IBM zum Thema: „System Netzwerk Architektur“

- Heute verbindet man aber im allgemeinen peer-to-peer mit den Bereichen die durch Napster (ca. 1998) in den Mittelpunkt rückten

Client-Server Model:

Klassisch:

- asymmetrisches Client Server System
  - leistungsfähiger und gut angebundener Server
  - läuft meistens über lange Zeit
  - stellt alle benötigten Ressourcen wie Speicher usw.
  - liefert Informationen/Dienste an eine gewisse Anzahl von Clients

  - Server ist der Flaschenhals für Leistung und Zuverlässigkeit

    Verringerung dieser Probleme durch serverseitige Lösungen:

    - Spiegelung der Daten
    - *load balancing*
    - Anfragen Routing

    Viele Server bestehen genau betrachtet aus mehreren einzelnen Maschinen.

Client-Server Model:

Neue Ideen/Entwicklungen:

- Ressourcen des Clients mit einbinden
- wurde zunehmend attraktiver:
  - Performanceunterschiede zwischen Server und Desktopsystemen verringerten sich
  - immer mehr Breitband Zugänge stehen zur Verfügung

Vom Client Server Model zu peer-to-peer:

- aus Client-Server Systemen die asymmetrischen Vorgaben entfernen
- jeder Client ist auch ein Server
- erlaubt Zugriff auf seine Ressourcen

- Clients sind nun echte *peers*
  - eigene Ressourcen werden im Tausch für das Nutzen des Services beigesteuert
- Die „Arbeit“ wird zwischen den *peers* aufgeteilt:
  - Nachrichten Weiterleitung
  - Berechnungen
  - Ablage/Speicherung
  - Suchen

Verhalten wie ein Server:

- eigene Ressourcen im Interesse anderer anbieten

Verhalten wie ein Client:

- andere *peers* im eigenem Interesse nach Ressourcen fragen

Die Wirklichkeit/Problematik:

- Das System bricht zusammen wenn nicht der überwiegende Teil der Nutzer bereit ist Ressourcen zur Verfügung zu stellen
- Vorgaben sind notwendig dieses System aufrecht zu erhalten

Zentraler Knoten/Performance Beschränkungen:

- im allgemeinen wird angenommen,
  dass peer-to-peer Systeme keinen zentralen (Haupt) Knoten besitzen
- dadurch:
    - sehr fehlertolerant
    - gute Performance

- jedoch haben viele Systeme trotzdem einen zentralen Knoten
- und haben Performance Beschränkungen

Performance Optimierung:

- Nutzung von Mittelsleuten, um die durchschnittliche Wartezeit zu verringern.
  Idee entspricht Proxy-Servern im Internet
- Arbeit möglichst im P2P System dynamisch verteilen
  (lokal orientiertes *load balancing*)
- Beispiel: *Content distribution systems*:

    - Anfrage bezogene Strategien um Daten zu verteilen
    - Daten an Knoten weiterleiten/speichern, die in der nähe des Knotens liegen, der die anfrage gestellt hat

    - ähnliche Ideen werden auch in Grid Systemen benötigt, um möglichst die nötigen Ressourcen in der Nähe von Datenquellen zu finden

Die „erste“ Software,

die peer-to-peer bekannt machte...

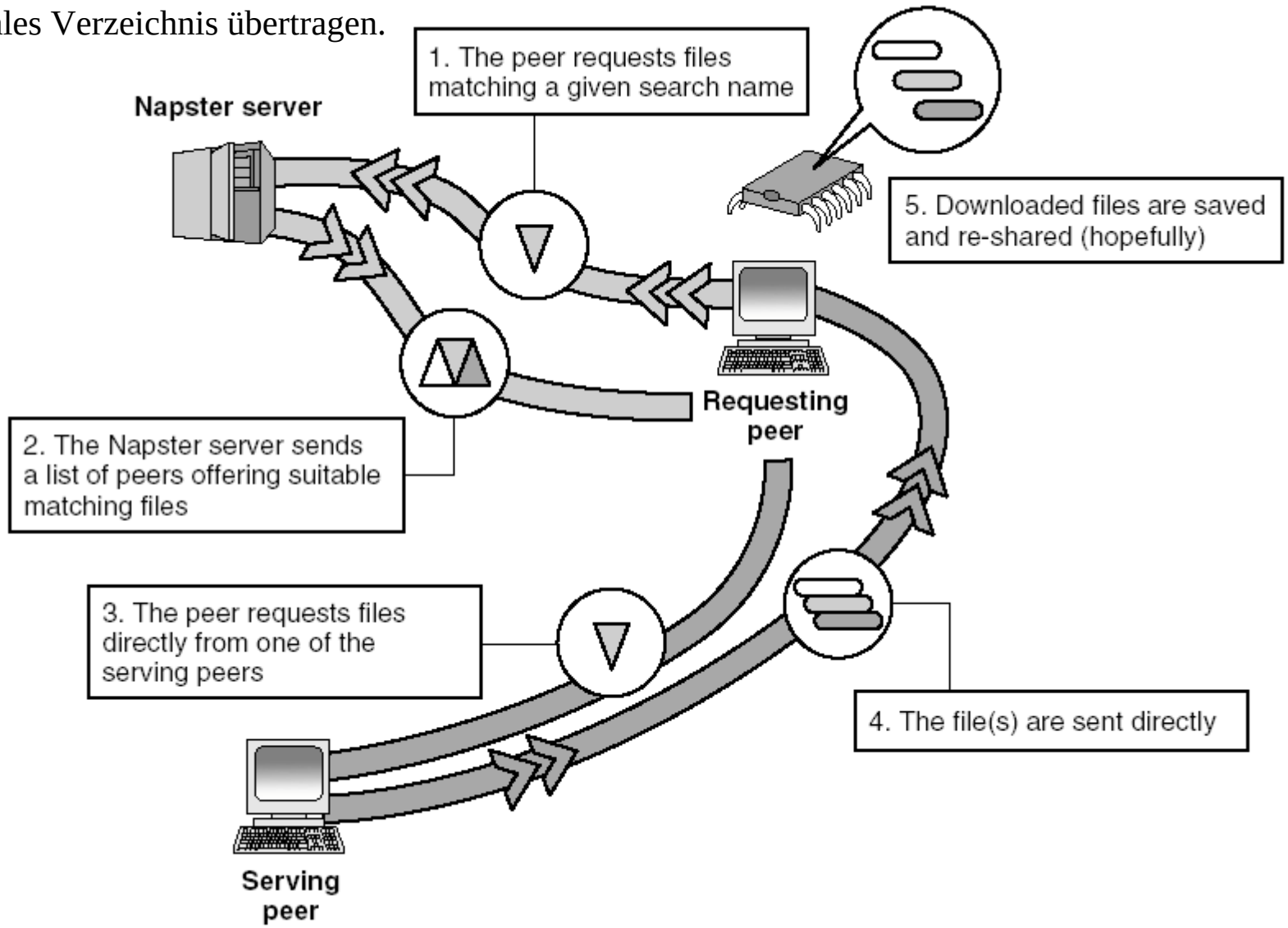
Napster:

- *file-sharing* System
- Suche und download von Musikstücken, die auf der Festplatte anderer Napster-User gespeichert ist

- wie funktioniert(e) Napster?
- echtes peer-to-peer?
- Probleme beim verwendeten System...
- Geschichtliches...

Wie funktioniert(e) Napster:

Bei Programmstart werden Daten über die zu tauschenden Musikstücke in ein globales Verzeichnis übertragen.



Jeder übernimmt die Client oder Server Rolle, abhängig ob download oder upload...

echtes peer-to-peer?

- unterschiedliche Meinungen
- Verzeichnisdaten, über die Musikstücke, liegen auf einem zentralen Server

- unabhängig davon:
    - Angebot hängt von einzelnen Nutzern ab
    - und nicht vom einzelnen Server

Probleme beim verwendeten System...

- technisches
  - litt unter einem einfachen Protokoll
  - schlechte Zuverlässigkeit und Bandbreite anderer Clients
  - schlechte oder abgebrochene Verbindungen behinderten oft den erfolgreichen erhalt von Musikstücken

Probleme beim verwendeten System...

- gesetzliche
  - das zentralisierte Verzeichnis wurde zum Flaschenhals, nicht nur technisch gesehen, sondern auch weil das zentralisierte System die Server anfällig für
    - gesetzliche
    - politische
    - und wirtschaftliche

    Angriffe machte

  - schließlich wurde Napster gerichtlich gestoppt

    (für die Unterstützung von Copyright Verletzungen)

Geschichtliches...

- unabhängig von technischen Problemen,
  erleichterte Napster auf drastische Weise die Möglichkeit
  Musikstücke kostenlos zu erhalten
- Napster machte peer-to-peer zu einem „Begriff“
- hatte an der Spitze bis zum 1.6 Millionen gleichzeitige Nutzer

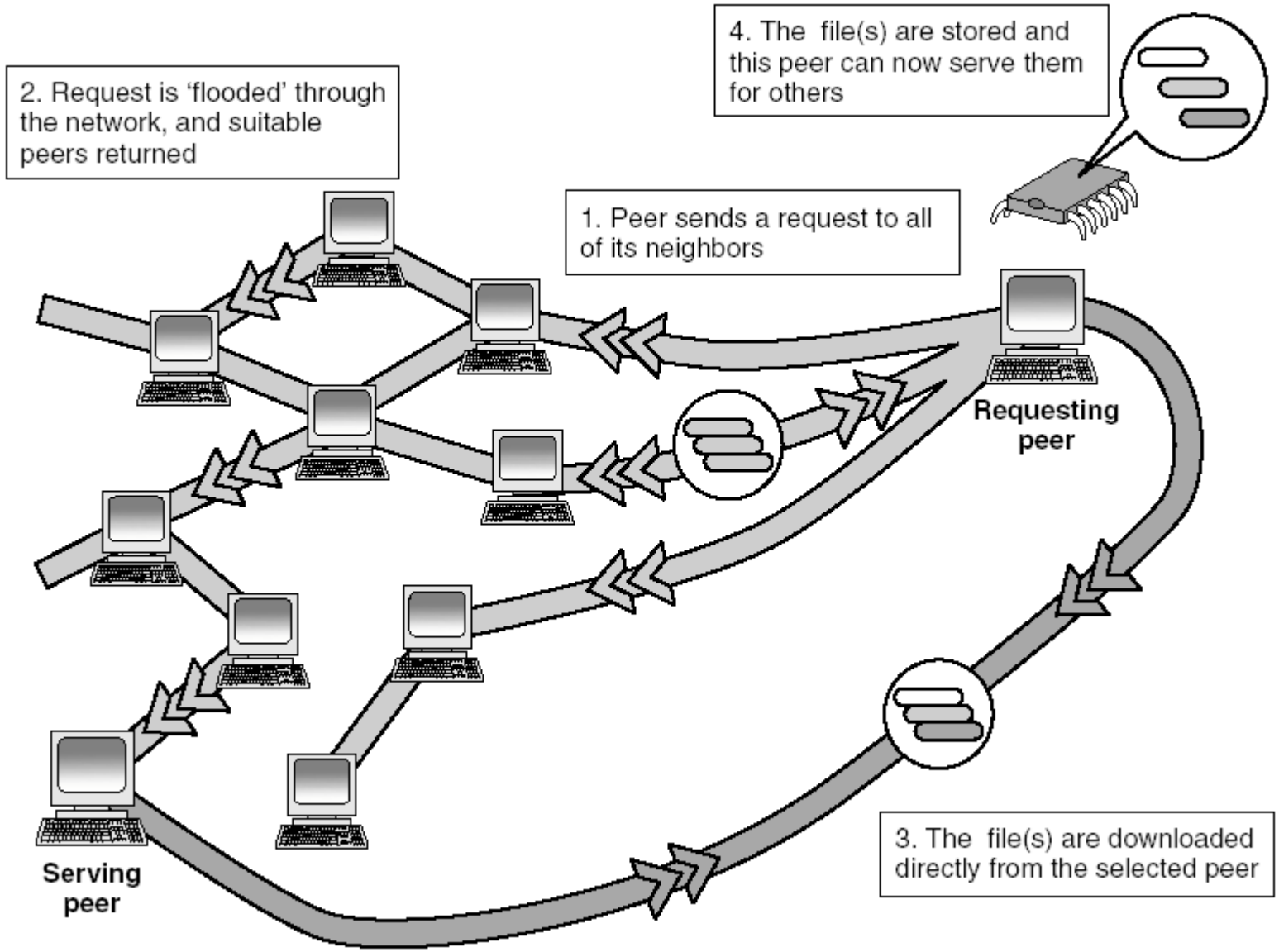
- demonstrierte als erste Software das ernorme Potential, dass in
  Client-Ressourcen steckt, die über einen Service
  angeboten/verbunden werden

Zukunft/Wunsch:

- Nach dem Niedergang von Napster entstand der Wunsch nach einem
  völlig dezentralisierten Service, um ähnlichen Angriffen widerstehen
  zu können
- Durch diesen Wunsch wurden viele wichtige Entwicklungen in
  verteilten Systemen vorangetrieben

Die 2. Generation...

- voll dezentralisiert
- distributed search protocol (Gnutella)
  - kein zentrales Verzeichnis notwendig
  - per broadcast werden Suchanfragen an Nachbarn verteilt


4. The file(s) are stored and this peer can now serve them for others
2. Request is 'flooded' through the network, and suitable peers returned
1. Peer sends a request to all of its neighbors
Requesting peer
Serving peer
3. The file(s) are downloaded directly from the selected peer

Probleme:

- trotz Begrenzung und Einschränkung von Suchabfragen ergaben Studien und Benutzererfahrungen:
  - verursachte Netzlast zu hoch
  - zu viele Anfragen und Steuernachrichten
  - dadurch Bandbreite für den eigentlich Transfer stark begrenzt

Abhilfe:

- Systeme für die Lokalisierung von Dateien/Inhalt (*Freenet*)
- Anfragen werden dahin geleitet, wo das Gesuchte voraussichtlich liegt
- Optimierung durch Einteilung der angebotenen Daten

- *Kazaa* und Neuentwicklungen von *Gnutella*
  - Einführung von *super-peers* (dynamisch)
  - speichern Anfragen
  - leiten Ergebnisse weiter

Die 3. Generation...

- effizientes Routing
- gesucht wurde ein zuverlässiger Weg
  - die Auslastung zu verteilen
  - benötigte Knoten zu Adressieren

- Schwerpunkte darauf gelegt:
  - das Kommen und Gehen von Knoten *beobachten* zu können
  - nur mit der *Kenntnis* eines Bruchteils des Netzwerkes arbeiten zu können
  - möglichst wenig *overhead* durch die Kommunikation

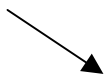
Effizientes Routing:

- Unterteilung
  - in Knoten fürs Nachrichten weiterleiten und
  - in Knoten für einen *Key-Location* Service
- Key-Location Service
  - populärste Ansätze setzen eine DHT (distributed hash table) ein
  - Knoten erhalten eine Pseudo-Zufällige Kennung
  - dadurch Position in einem Schlüsselraum festgelegt
  - Nachrichten erhalten einen ähnlichen Pseudo-Zufälligen Schlüssel
  - dieser Schlüssel bestimmt welcher Knoten für diese Nachricht zuständig ist
  - Nachrichten im eigenem Schlüsselraum werden selbst verteilt
  - Nachrichten für einen anderen Schlüsselraum werden direkt an diesen weitergeleitet
- Die Möglichkeit Routing-Services anbieten zu können, führte zu einem großen Anstieg von P2P Anwendungen

Weitere Optimierungen:

- Swarm distribution (Kazaa)
  - Verbessert das load balancing und reduziert die Abhängigkeit von einzelnen Knoten,
    durch das Abfragen des secure hash Schlüssels einer Datei,
    statt dem abfragen des Dateinamens
  - Mit diesem Schlüssel ist es möglich abgebrochene Übertragungen fortsetzen zu können

Transfer bricht ab



Die Datei wird bei anderen Knoten über den secure hash Schlüssel gesucht

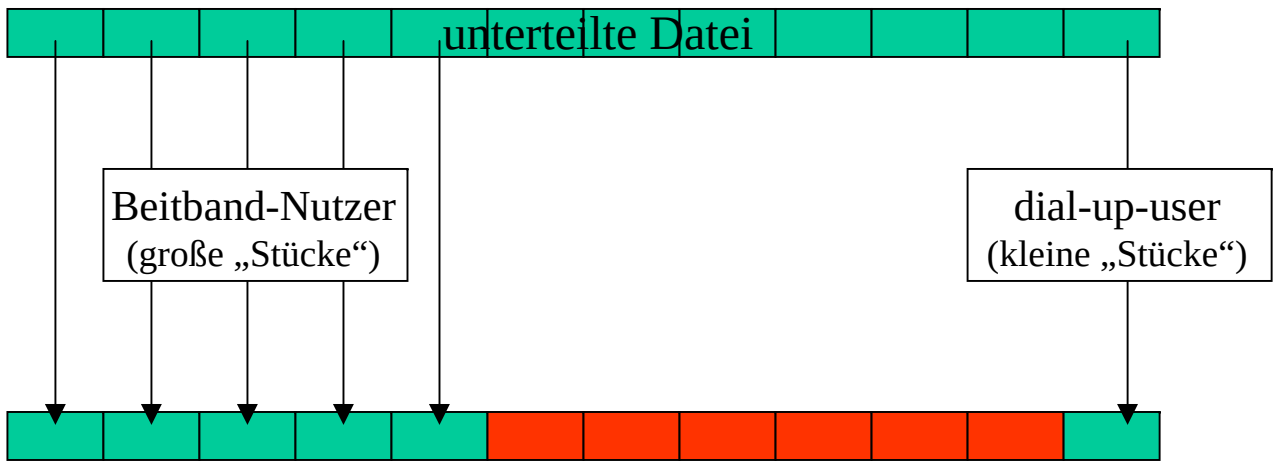
dieselbe Datei wird gefunden

Transfer wird wieder aufgenommen

Weitere Optimierungen:

- Unterteilung der Dateien in kleinere Einzelstücke
  - Übertragung von mehreren Knoten möglich
  - Schnellerer erhalt der vollständigen Datei
  - Teilstücke können bereits wieder mit verteilt werden

  - weitere Möglichkeit:



... jeder kann seinen Teil dazu beitragen

Weitere Optimierungen:

- effektive Verteilung
- Kopien von angeforderten Daten werden in der „Nähe“ des Nutzers verteilt, dadurch Optimierung beim Übertragen
- wird von Streaming Anbietern in ähnlicher Weise genutzt

- Das Erreichen von deterministischen Ergebnissen bei Suchanfragen ist in völlig dezentralen Umgebungen stark begrenzt

Deshalb werden z.B. Einschränkungen beim *Angebot* gemacht
... nur Musik, nur Software ...
um den Suchbereich einzuschränken.

Anwendungen:

- *Eternity*
  - System für die dauerhafte Ablage von Daten
  - Daten werden in einem P2P ähnlichem System verteilt
  - mehrfach wiederholt gespeichert
  - dadurch ausfallsicher
  - und gleichzeitige Optimierung der Erreichbarkeit
- *Free Haven* (http://www.freehaven.net/)
  - Dokument Speicher System (nutzt Eternity)
  - anonymes Speichern von Dokumenten

Was ist zu beachten ?

- jeder Teilnehmer soll möglichst gleich belastet werden
- bei der Nutzung von DHT‘s die Anzahl der Schlüssel
  gleichmäßig verteilen,
  damit jeder Knoten gleichmäßig durch Anfragen belastet wird
- Nutzer haben oft schlechte,
  vor allem aber sehr unterschiedliche Anbindungen
- dadurch Kommunikation zwischen den Knoten begrenzt
- einige User können keine eingehenden Verbindungen akzeptieren
  (bedingt durch Firewall, Router, ...)
- Breitbandzugänge haben unterschiedliche up- und downstreams
- schlechte bis gar keine Administrierbarkeit
  (z.B. Verbot von bestimmten Dateien...)
- ständiges *kommen* und *gehen* von Knoten
- Performance und Verlässlichkeit beim empfangen von Dateien
  hängt jeweils vom anderen Knoten ab
- dadurch ist jede Angabe von *quality-of-service* nicht oder schwer möglich

Die 4. Generation ???

Die 4. Generation ???

- viele Interessante Ansätze für neuen Routing-Methoden vorhanden
- sehr überzeugend in groß Simulationen
- viele aber noch nicht unter reellen Bedingungen getestet