



Systeme II

**12. Vorlesungswoche
14.07. – 18.07.2008**

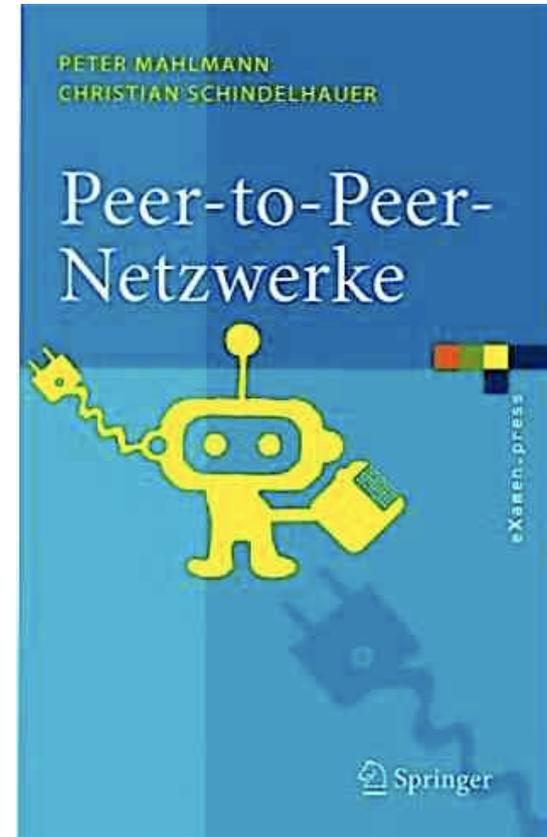
Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Rechnernetze und Telematik
Christian Schindelhauer
Sommer 2008

Peer-to-Peer-Netzwerke

▶ Peer-to-Peer-Netzwerke

- Peter Mahlmann, Christian Schindelhauer
Peer-to-Peer-Netzwerke - Methoden und Algorithmen
- Springer Berlin
- 293 Seiten
- über 120 Abbildungen

▶ **32,95 Euro**



Meilensteine Praxis

- ▶ **Napster (1999)**
 - seit 1999, bis 2000 (Gerichtsurteil)
- ▶ **Gnutella (2000)**
 - Neue Version (Gnutella 2) in 2002
- ▶ **Edonkey (2000)**
 - Später: Overnet unter Verwendung von Kademlia
- ▶ **FreeNet (2000)**
 - Anonymisierung der Teilnehmer
- ▶ **JXTA (2001)**
 - Open Source Peer-to-Peer-Netzwerk-Plattform
- ▶ **FastTrack (2001)**
 - bekannt durch KaZaa, Morpheus, Grokster
- ▶ **Bittorrent (2001)**
 - Nur Download-System, keine Suche
- ▶ ...

Meilensteine Theorie

- ▶ **Distributed Hash-Tables (DHT) (1997)**
 - Urspr. für Lastverteilung zwischen Web-Servern
- ▶ **CAN (2001)**
 - Effiziente verteilte DHT-Datenstruktur für P2P-Netzwerke
- ▶ **Chord (2001)**
 - Effiziente verteilte P2P-Datenstruktur mit logarithmischer Suchzeit
- ▶ **Pastry/Tapestry (2001)**
 - Effiziente verteilte P2P-Datenstruktur aufbauend auf Routing von Plaxton
- ▶ **Kademlia (2002)**
 - P2P-Lookup basierend auf XOR-Metrik
- ▶ **Viele weitere interessante Netzwerke**
 - Viceroy, Distance-Halving, Koorde, Skip-Net, P-Grid, ...

Napster

- ▶ **Shawn (Napster) Fanning**
 - brachte Juni 1999 eine Beta-Version seines mittlerweile legendären Napster-Peer-to-peer-Netzwerks heraus
 - Ziel: File-sharing-System
 - Tatsächlich: Musik-Tauschbörse
 - Herbst 1999 war Napster Download des Jahres
- ▶ **Urheberrechtsklage der Musik-Industrie im Juni 2000**
- ▶ **Schließung von Napster im Jahr 2000**

Wie funktioniert Napster?

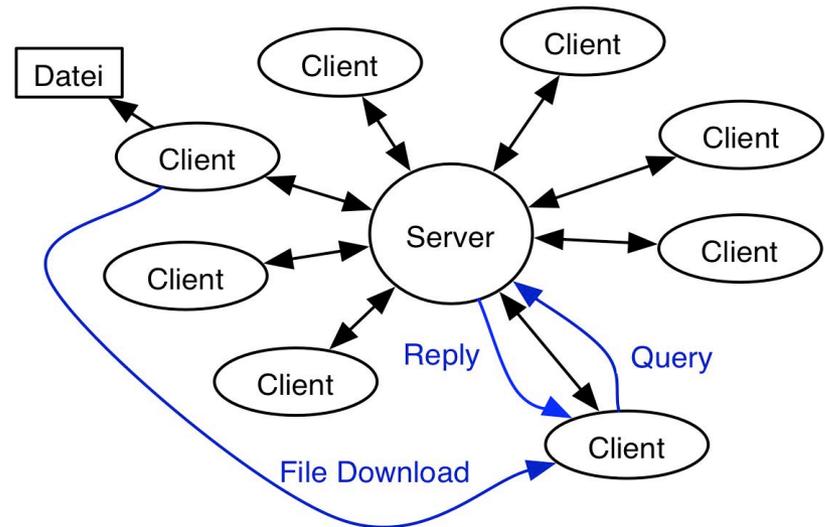
➤ Client-Server-Struktur

➤ Server unterhält

- Index mit Meta-Daten
 - Dateiname, Datum, etc
- Tabelle der Verbindungen der teilnehmenden Clients
- Tabelle aller Dateien der teilnehmenden Clients

➤ Query

- Client fragt nach Dateinamen
- Server sucht nach passenden Teilnehmern
- Server antwortet, wer die Datei besitzt
- Anfrage-Client lädt Datei von datei-besitzenden Client herunter



Wie gut ist Napster?

▶ Vorteile

- Napster ist einfach
- Dateien werden schnell und effizient gefunden

▶ Nachteile

- Zentrale Struktur erleichtert Zensur, feindliche Eingriffe und technisches Pannen
 - wie z.B. Denial-of-Service-Angriff
- Napster skaliert nicht
 - d.h. mit zunehmender Teilnehmerzahl verschlechtert sich die Performanz
 - Speicher auf dem Server endlich

▶ Resumee

- Napster keine akzeptable Peer-to-Peer-Netzwerklösung
- Bis auf den Download-Aspekt ist Napster im eigentlichen Sinne kein P2P-Netzwerk

Gnutella - Geschichte

▶ Gnutella

- wurde im März 2000 herausgegeben von Justin Frankel und Tom Pepper von Nullsoft
- Nullsoft ist seit 1999 eine Tochter von AOL

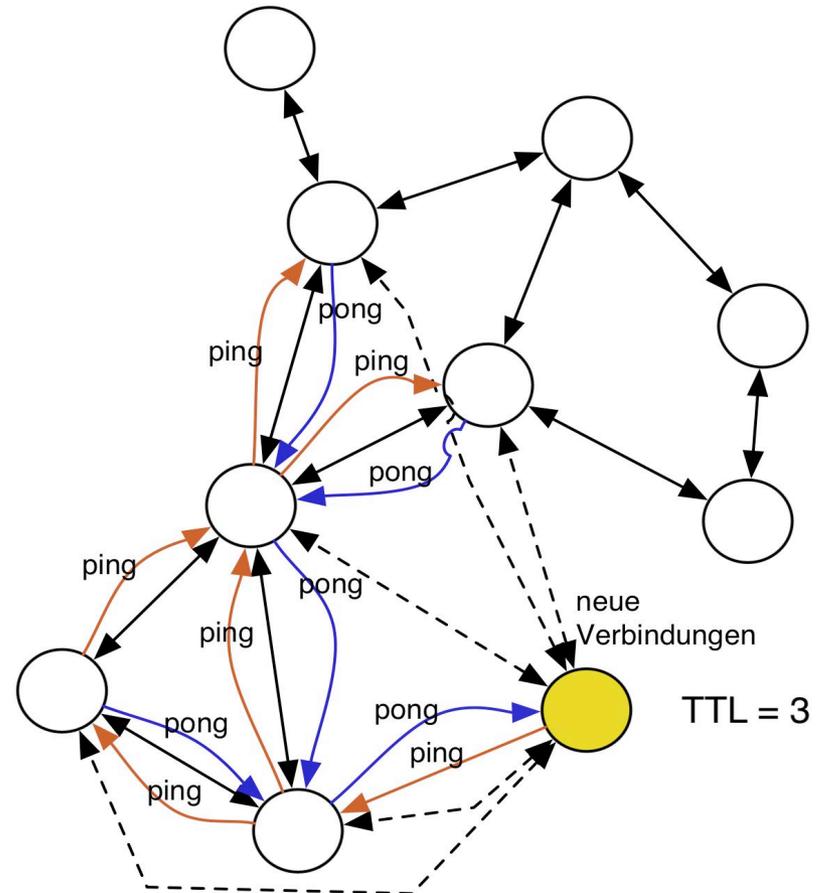
▶ File-Sharing-System

- Ziel wie Napster
- Arbeitet aber völlig ohne zentrale Strukturen

Gnutella - Originalversion - Anbindung

► Nachbarschaftslisten

- Gnutella verbindet direkt mit anderen Clients
- Beim Download wird eine Liste von Clients mitgeliefert
- Diese werden ausprobiert bis ein Aktiver sich meldet
- Ein aktiver Client gibt dann seine Nachbarschaftsliste weiter
- Nachbarschaftslisten werden immer weiter verlängert und gespeichert
- Die Anzahl aktiver Nachbarn ist beschränkt (typisch auf fünf)



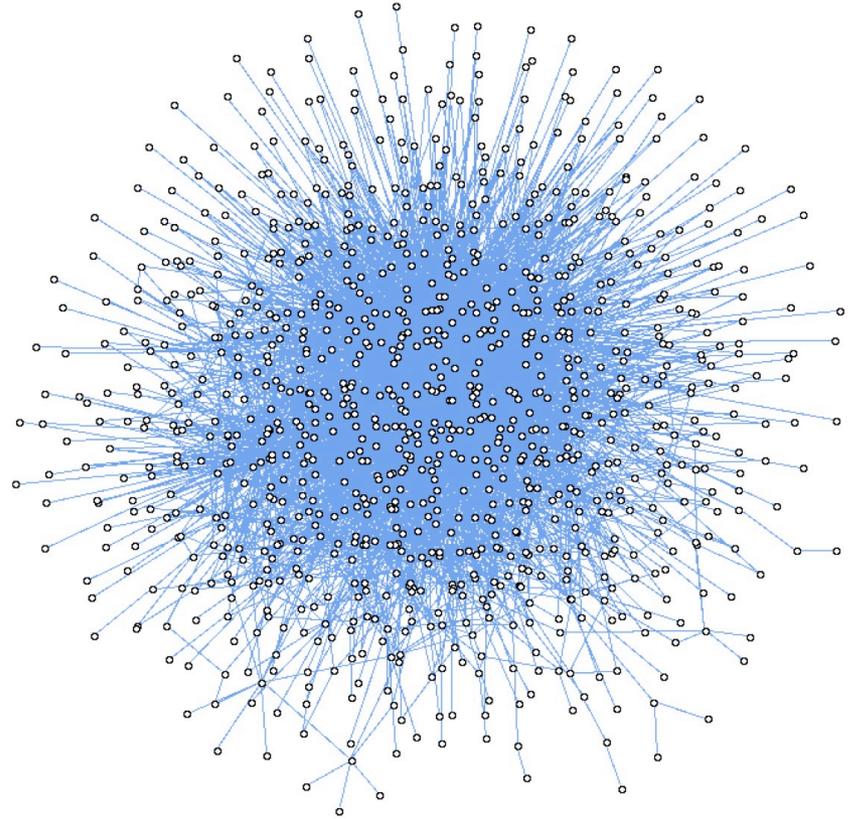
Gnutella - Originalversion - Anbindung

► Protokoll

- Ping
 - Teilnehmeranfrage
 - werden weiter gereicht gemäß TTL-Feld (time to live)
- Pong
 - Reaktion auf Ping
 - Werden auf dem Anfragepfad zurückgereicht
 - IP und Port des angefragten Teilnehmers
 - Anzahl und Größe zur Verfügung gestellter Dateien

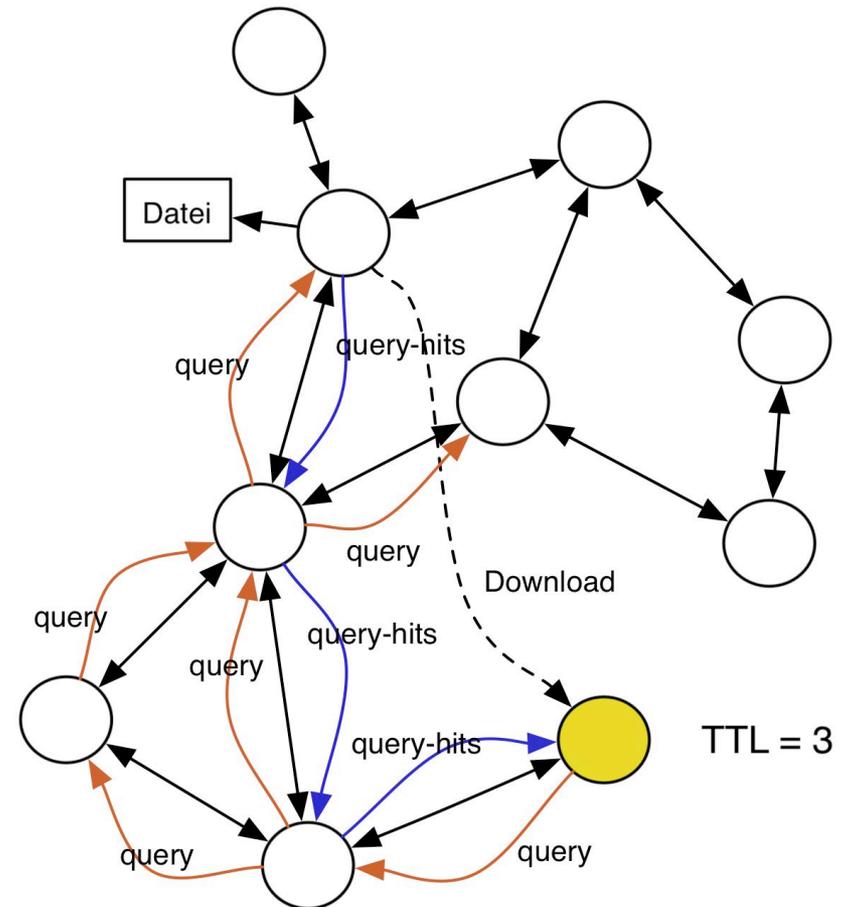
► Graphstruktur

- entsteht durch zufälligen Prozess
- unterliegt Pareto-Verteilung
- entsteht unkontrolliert



Gnutella - Originalversion - Anfrage

- ▶ **Dateianfrage**
 - wird an alle Nachbarn geschickt
 - diese senden sie an ihre Nachbarn
 - bis zu einer vorgegebenen Anzahl von Hops
 - TTL-Feld (time to live)
- ▶ **Protokoll**
 - Query
 - Anfrage nach Datei wird bis zu TTL-hops weitergereicht
 - Query-hits
 - Antwort auf umgekehrten Pfad
- ▶ **Wenn Datei gefunden wurde, direkter Download**



Gnutella - Diskussion

▶ Vorteile

- verteilte Netzwerkstruktur
- Netzwerk skalierbar

▶ Nachteil

- Durch TTL findet für Abfragen eine implizite Netzwerkpartitionierung statt
- Dadurch Anfrageerfolg gering
- Durch lange Wege, große Latenzzeiten

Distributed Hash-Table (DHT)

▶ Hash-Tabellen

▶ Vorteile

- Suche einfach

▶ Nachteile

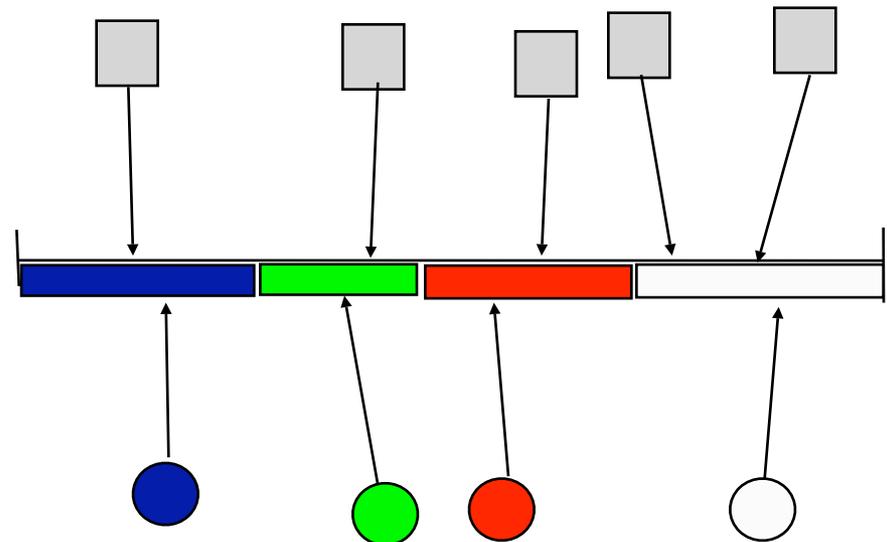
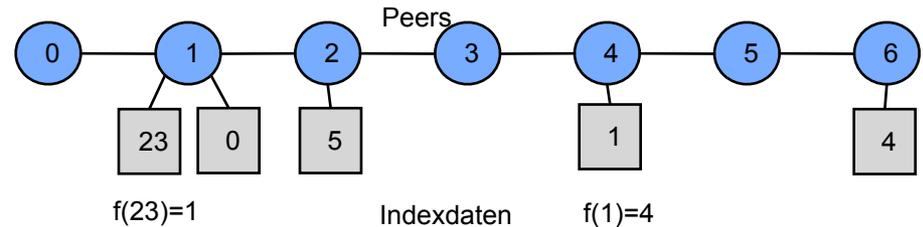
- Ein neuer Peer verursacht neue Wahl der Hash-Funktion
- Lange Wege

▶ Distributed Hash-Table

▶ Peers werden an eine Stelle ge“hash“t und erhalten Bereiche des Wertebereichs der Hashfunktion zugeteilt

▶ Daten werden auch ge“hash“t

- Je nach Bereich den Peers zugeordnet



Einfügen in die Distributed Hash-Table (DHT)

▶ Distributed Hash-Table

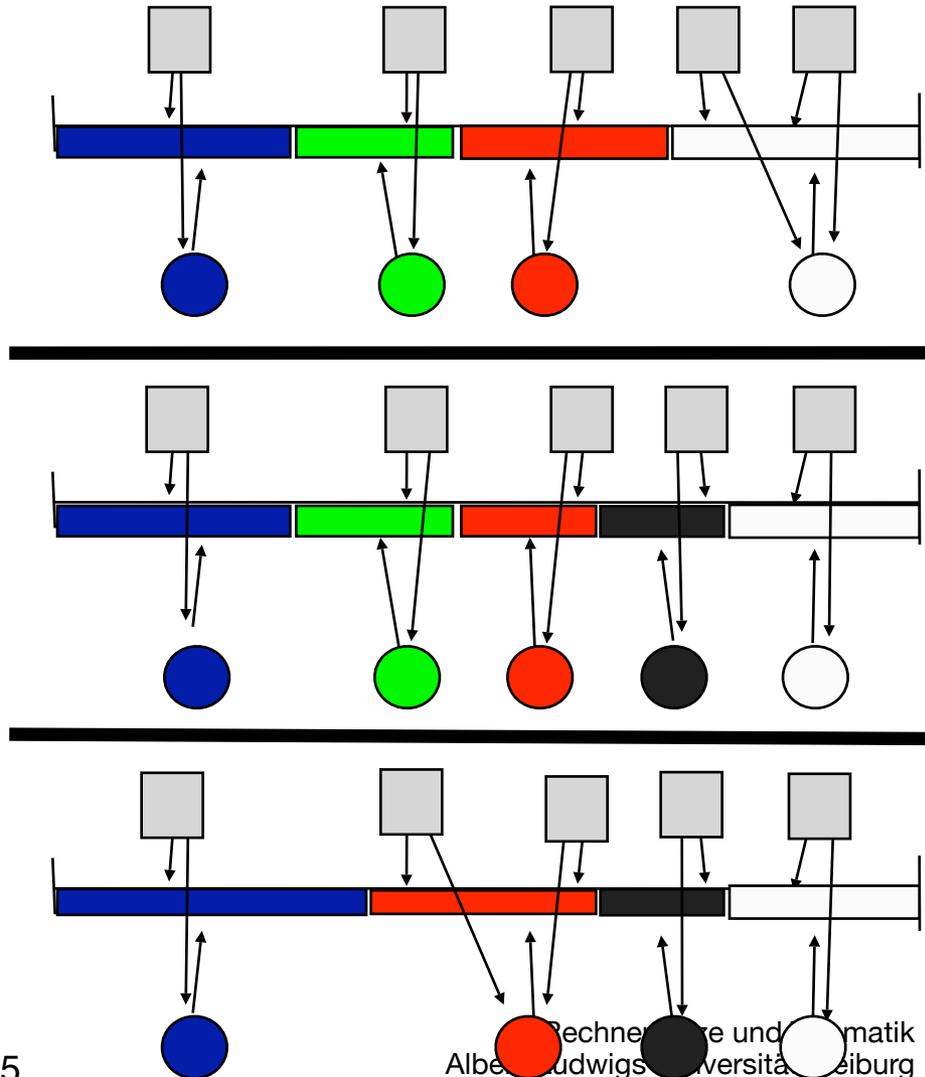
- Peers werden an eine Stelle ge“hash“t
- Dokumente ebenso
- Jeder ist für einen Bereich verantwortlich

▶ Kommt ein neuer Knoten hinzu

- müssen die Nachbarn teilen

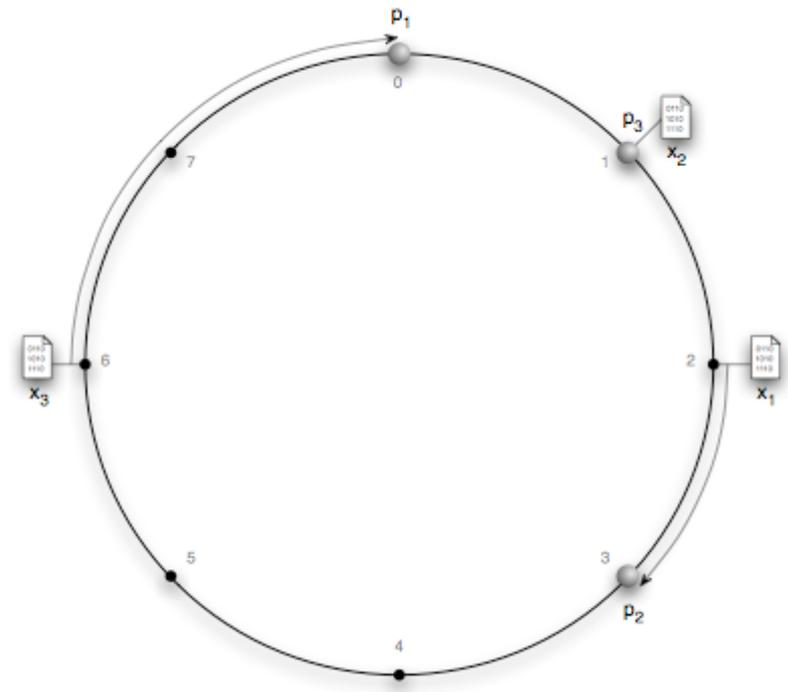
▶ Verlässt ein Knoten das Netzwerk

- übernehmen die Nachbarn sein Gebiet



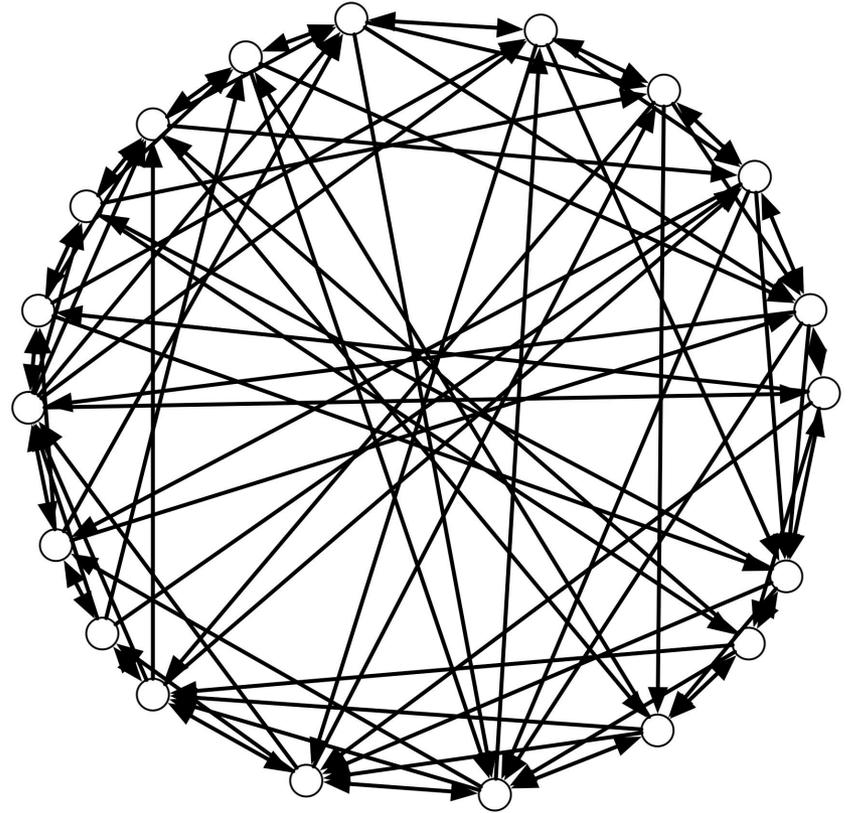
Chord als DHT

- ▶ **Zwei Hash-Funktionen bilden Peers und Daten auf den Ring ab**
 - Das Datum wird dem nächsten Peer zugewiesen



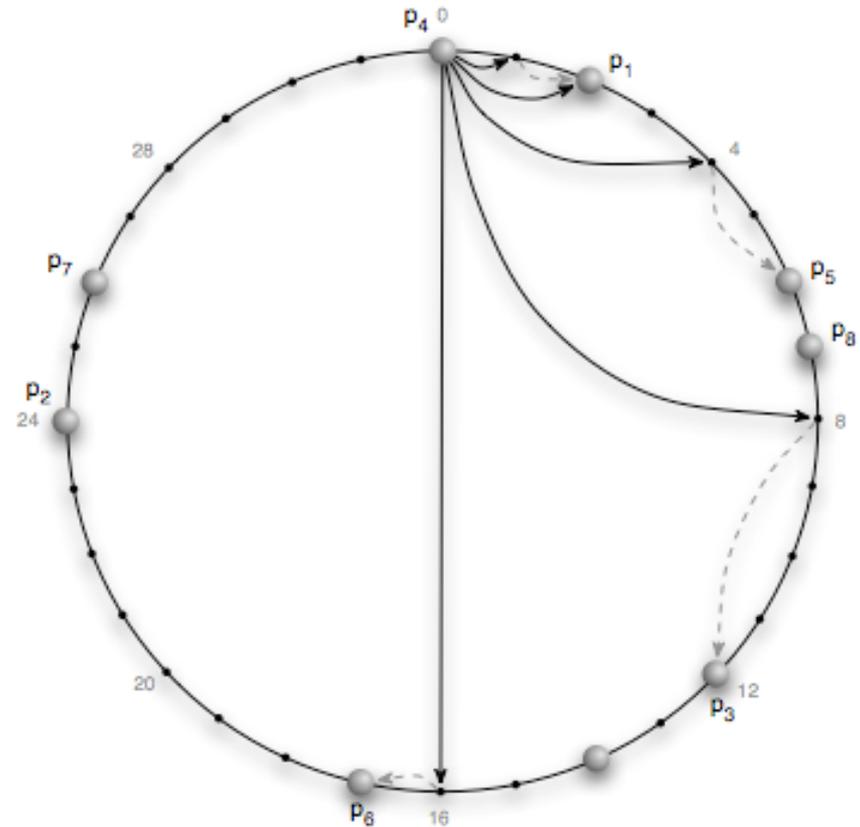
Chord

- ▶ von Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek und Hari Balakrishnan (2001)
- ▶ DHT mit Hash-Bildbereich $\{0, \dots, 2^m - 1\}$
 - für genügend großes m
- ▶ Ring-Verknüpfung der Peers
- ▶ Abkürzungen im Ring durch exponentiell gestaffelte Zeiger auf Nachfolger



Die Datenstruktur von Chord

- ▶ **Für jeden Knoten b :**
 - successor: Nachfolger
 - predecessor: Vorgänger
 - Für $i \in \{0, \dots, m-1\}$
 - $\text{Finger}[i] :=$ Der Knoten der dem Wert $r_v(b+2^i)$ folgt
- ▶ **Für kleine i werden die Finger-Einträge immer gleich**
 - Nur unterschiedliche Fingereinträge werden gespeichert
- ▶ **Lemma**
 - Die Anzahl unterschiedlicher Finger-Einträge für Knoten b ist mit hoher Wahrscheinlichkeit $O(\log n)$



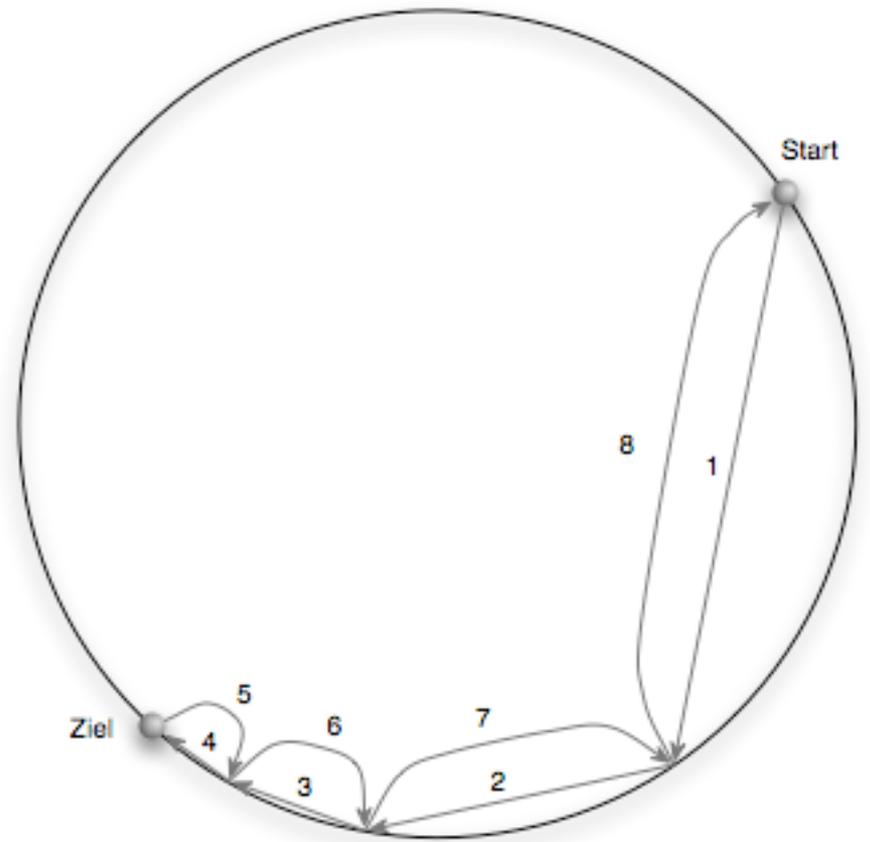
Suchen in Chord

▶ Theorem

- Die Suche braucht mit hoher W'keit $O(\log n)$ Sprünge

▶ Beweis:

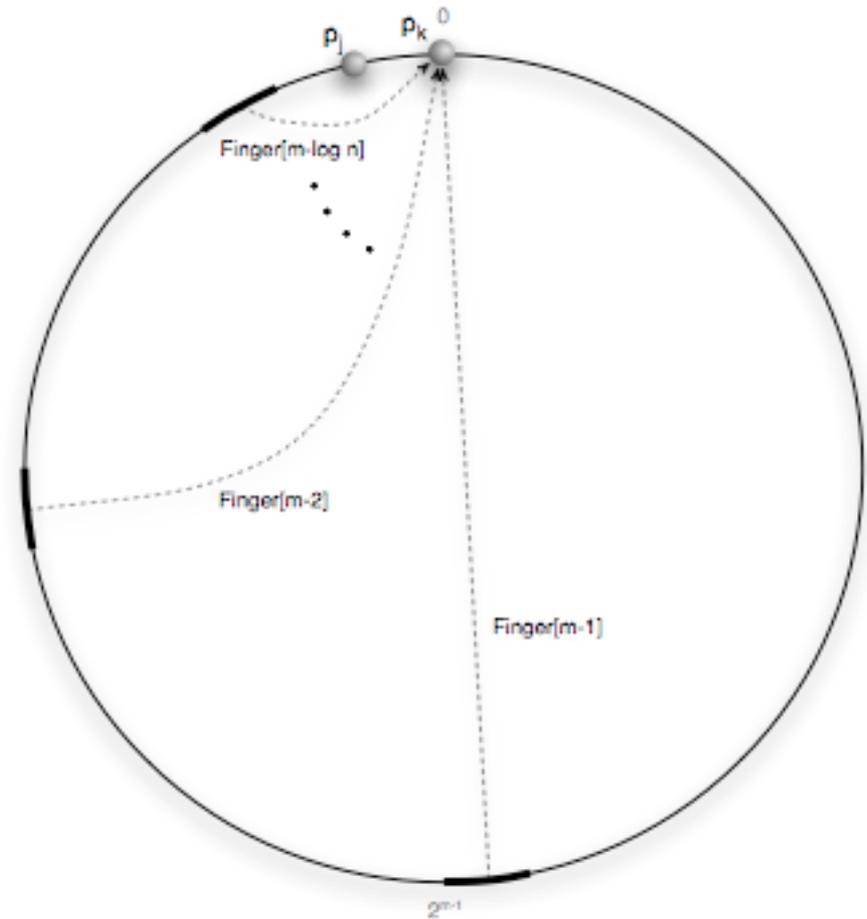
- Mit jedem Sprung wird die Entfernung zum Ziel mindestens halbiert
- Zu Beginn ist der Abstand höchstens 2^m
- Der Mindestabstand zweier benachbarter Peers ist $2^m/n^c$ mit hoher W'keit
- Damit ist die Laufzeit beschränkt durch $c \log n$



Fingeranzahl

▶ Lemma

- Der Ausgrad im CHORD-Netzwerk ist $O(\log n)$ mit hoher W' keit
- Der Eingrad im CHORD-Netzwerk ist $O(\log^2 n)$ mit hoher W' keit



IP Multicast

► Motivation

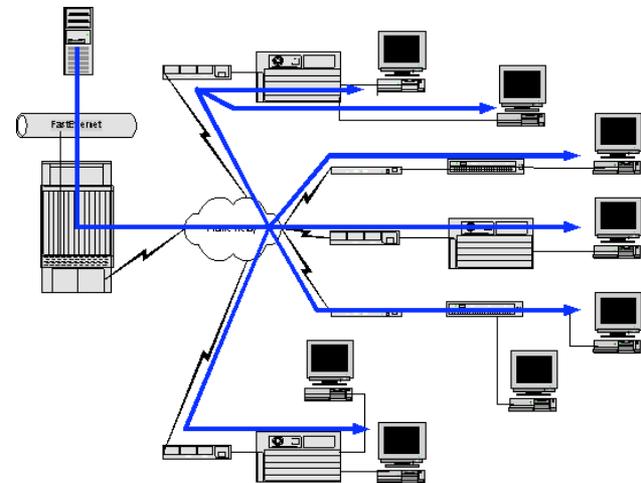
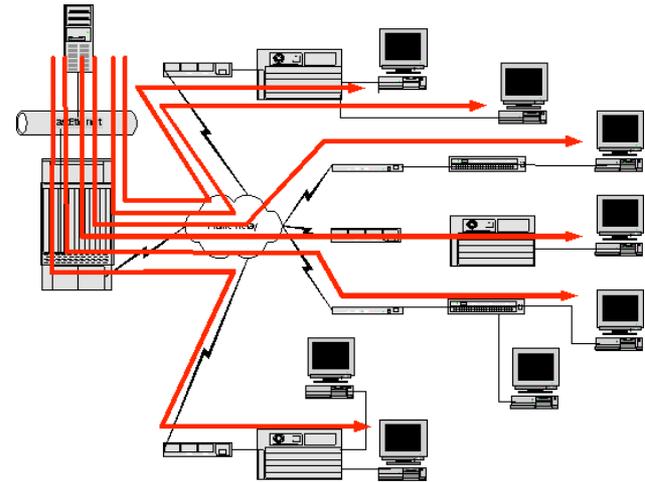
- Übertragung eines Stroms an viele Empfänger

► Unicast

- Strom muss mehrfach einzeln übertragen werden
- Bottleneck am Sender

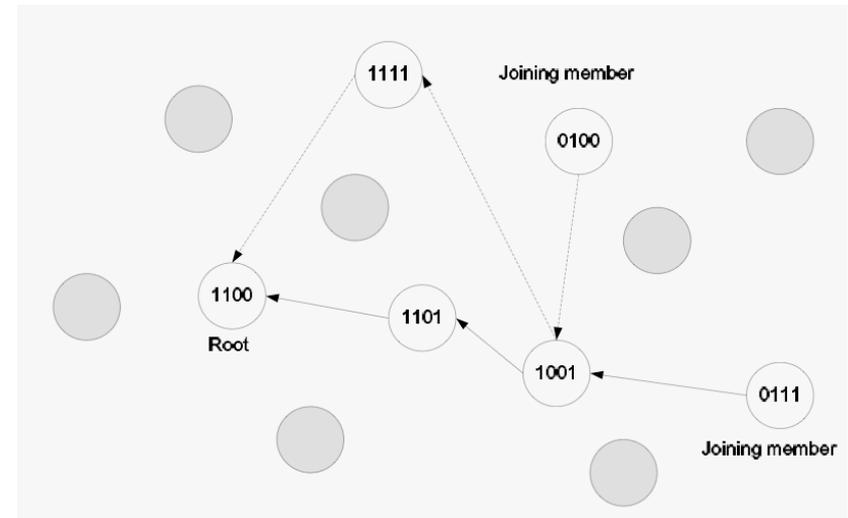
► Multicast

- Strom wird über die Router vervielfältigt
- Kein Bottleneck mehr



Scribe

- ▶ **Multicast-Baum im Overlay-Netzwerk**
- ▶ **Scribe [2001] basiert auf Pastry**
 - Castro, Druschel, Kermarrec, Rowstron
- ▶ **Vergleichbare Ansätze**
 - CAN Multicast [2001] basiert auf CAN
 - Bayeux [2001] basiert auf Tapestry



Funktionsweise Scribe

▶ Create

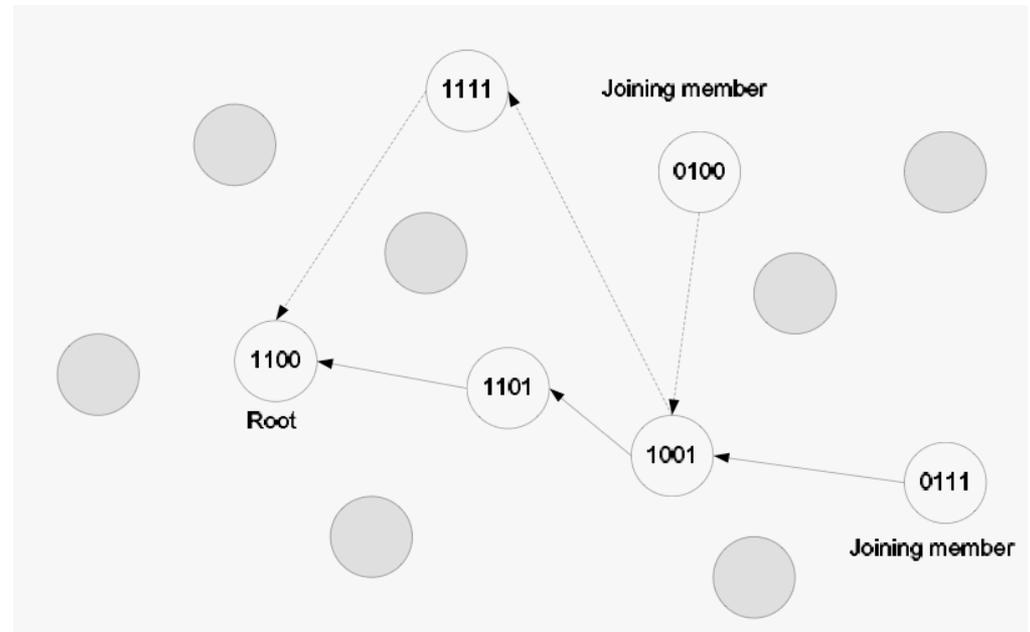
- GroupID wird einem Peer gemäß nächsten Pastry-Index zugewiesen

▶ Join

- Interessierter Peer macht Look-up zur Group-ID
- Sobald ein Peer im Multicast-Baum gefunden worden ist, wird neuer Teilpfad eingefügt

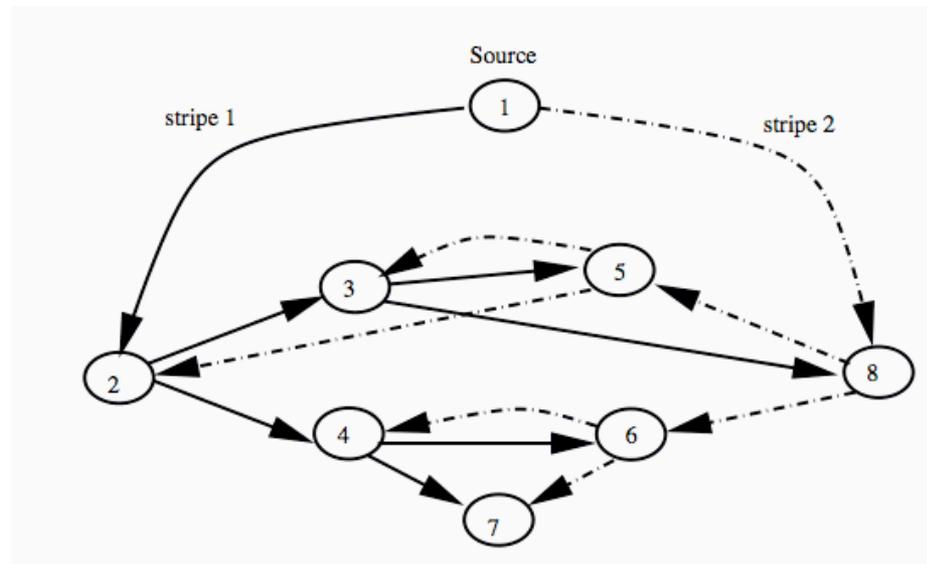
▶ Download

- Nachrichten werden baumförmig verteilt
- Knoten duplizieren Teile



Split-Stream

- ▶ **Castro, Druschel, Kermarrec, Nandi, Rowstron, Singh 2001**
- ▶ **Idee**
 - Teile die Datei der Größe B in k kleinere Teile
 - Verwende anderen Multicast-Baum für jeden der Teile
 - Dadurch wird jeder Peer mal als Blatt oder als Verteil-Knoten fungieren
 - außer der Quelle
- ▶ **Der Upload jedes Knotens ist dann (im Idealfall) höchstens der Download**



Bittorrent

- ▶ **Bram Cohen**
 - ▶ **Bittorrent ist ein reales (sehr erfolgreiches) Peer-to-Peer-Netzwerke**
 - konzentriert sich auf Download
 - verwendet (implizit) Multicast-Trees für die Verteilung der Daten
 - ▶ **Beschreibung ist Peer-orientiert und nicht Daten-orientiert**

 - ▶ **Ziele:**
 - Möglichst effizienter Download einer Datei unter Zuhilfenahme der Upload-Fähigkeit der Peers
 - Möglichst effiziente Ausnutzung des Upload von Peers
- In der Praxis ist der Upload der Bottleneck
 - z.B. wegen der asymmetrischen Protokoll-Gestaltung von ISDN, Bitübertragungsschicht von DSL
 - Fairness zwischen den Peers
 - Seeders versus Leechers
 - Verwendung verschiedener Quellen

Bittorrent

Koordination und Datei

▶ **Zentrale Koordination**

- durch sogenannten Tracker-Host
- Für jede Datei gibt der Tracker eine Menge von Zufallspeers aus der Menge der downloadenden Peers
- Zusätzlich: Ausgabe des Hash-Codes und anderer Kontroll-Information
- Tracker-Hosts haben keine Dateien
 - Trotzdem kann das Anlegen einer Tracker-Datei auf einem Tracker-Host rechtliche Probleme ergeben (Urheberrechtsgesetz)

▶ **Datei**

- ist in kleinere Dateien zerlegt (in Tracker-Datei festgelegt)

- Jeder teilnehmende Host kann heruntergeladenen Teil weiterverbreiten, sobald er vollständig erhalten wurde
- Damit ist Bittorrent die Umsetzung eines Split-Stream-ähnlichen Protokolls

▶ **Interaktion zwischen Peers**

- Zwei Peers tauschen die Information über ihre vorhandenen Teile aus
- Gemäß der Politik von Bittorrent werden dann noch nicht vorhandene Teile von dem einen Peer zum anderen übertragen

Bittorrent Politik

- ▶ **Ziel**
 - Selbstorganisierendes System
 - Gute (Uploader/Seeder) werden belohnt
 - Böse (Downloader/Leecher) werden bestraft
- ▶ **Belohnung**
 - Gute Download-Bandweite
 - Rücknahme einer Drosselung (un-choke)
- ▶ **Bestrafung**
 - Drosselung der Bandweite (choke)
- ▶ **Bewertung**

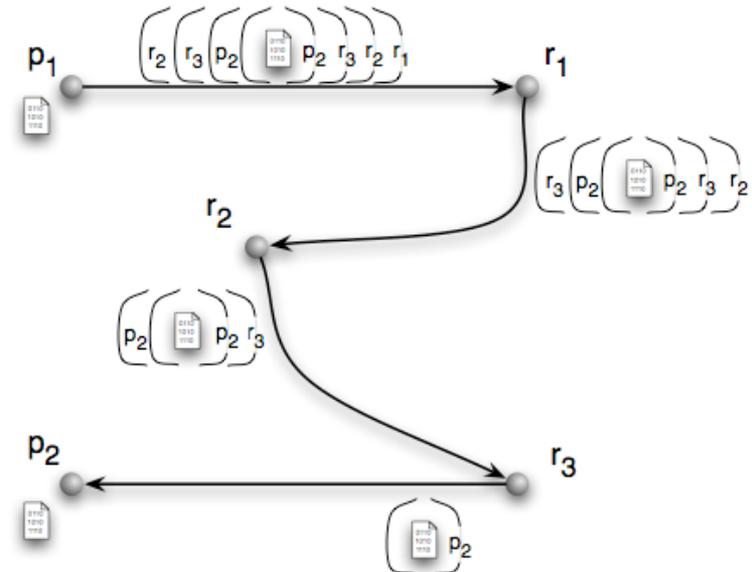
Methoden der Anonymisierung

- ▶ **Dining Cryptographers**
 - Wer hat's geschickt?
- ▶ **Onion Routing**
 - Verwickelte Umwege...
- ▶ **F2F-P2P**
 - Friend-to-Friend
- ▶ **Dark-Net**
 - War das was?
- ▶ **Steganographie**
 - nichts zu sehen...
- ▶ **k-aus-n-Verschlüsselung**
- ▶ **Verschlüsselte Inhalte**
 - Denn sie wissen nicht, was sie speichern...
- ▶ **Verschlüsselte, unterschriebene Index-Einträge**
 - gezeichnet: Zorro

Onion Routing

- ▶ **Von David Goldschlag, Michael Reed, and Paul Syverson**
- ▶ **Ziel**
 - Schutz der Privatsphäre von Sender und Empfänger einer Nachricht
 - Schutz der übermittelten Nachricht
- ▶ **Annahme**
 - Spezielle Infrastruktur (Onion Routers), die bis auf wenige Ausnahmen kooperieren
- ▶ **Methode:**
 - Basierend auf Mix Cascades (D. Chaum)
 - Nachricht wird von der Quelle zum Ziel über Zwischenstationen geleitet (Proxies - Onion Routers)
 - Onion Routers wählen unvorhersehbar andere Onion Router als Zwischenstationen
 - Zwischen Sender, Onion Routers und Empfängern ist die Nachricht jeweils symmetrisch verschlüsselt

- Jeder Onion Router kennt nur die nächste Zwischenstation
 - Die Nachricht ist wie eine Zwiebel mehrfach für die Zwischenstationen verschlüsselt
- ▶ **Onion Routers sind eine freiwillige Infrastrukturerweiterung des Internets**
- Verstehen sich nicht als Peer-to-Peer-Netzwerk



Free-Net

▶ Netzwerkstruktur

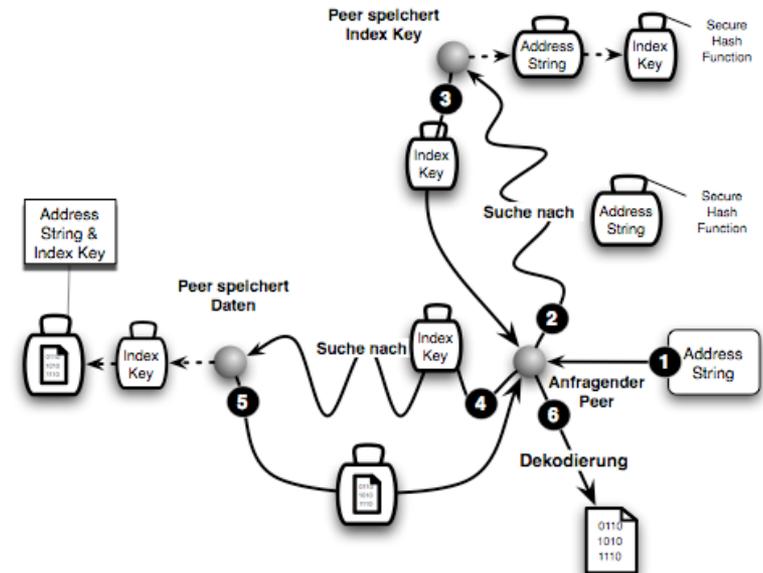
- stark verwandt mit Gnutella
- Netzwerkaufbau durch Nachbarkanten
 - aber kein F2F-Netzwerk, da bei der Suche Abkürzungen eingebaut werden können
- Ähnlich wie Gnutella ist das Netzwerk Pareto-verteilt

▶ Speichern von Dateien

- Jede Datei kann durch den kodierten Adress-String und dem signierten Index-Schlüssel (signed subspace key) gefunden, entschlüsselt und gelesen werden
- Jede Datei wird mit der Information des Index-Schlüssels gespeichert, aber ohne kodierten Adress-String
- Dadurch kann kein Server diese Datei lesen
 - es sei denn er führt eine Wörterbuch-Attacke durch

▶ Speichern von Index-Daten

- Der Adress-String, kodiert durch eine kryptographische Hash-Funktion führt zu den passenden Peer, der die Index-Daten bestehend aus dem Adress-String und dem signierten Index-Schlüssel besteht
- Mit diesen Index-Daten kann die Datei gefunden werden



FastTrack & Gnutella2

► Hybride Struktur

- Knoten mit großer Bandbreite werden zu P2P-Server ausgewählt
- Diese unterhalten P2P-Netzwerk im Stil von Gnutella
- Normale Knoten werden als Clients an diese Super-Knoten angebunden

► Eingesetzt in

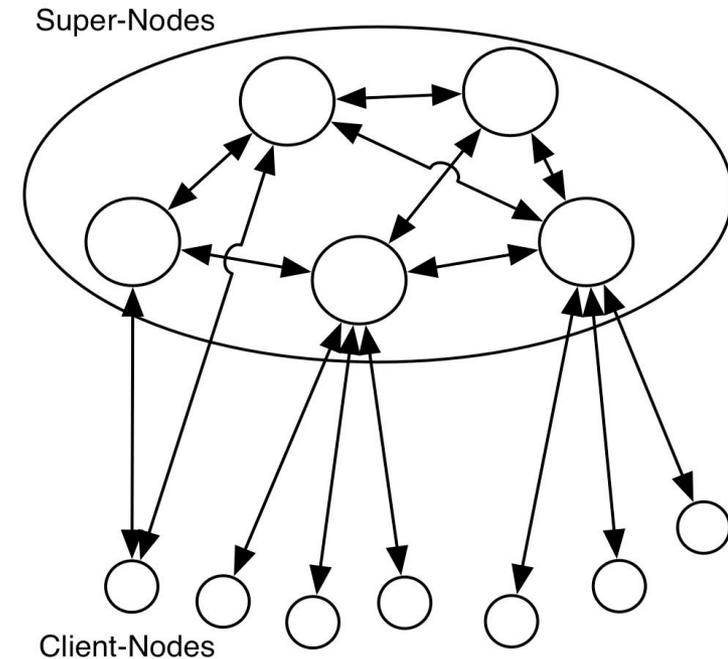
- FastTrack
- Gnutella2 (neuere Ausgabe)

► Vorteile

- Verbesserte Skalierbarkeit
- Geringere Latenzzeiten

► Nachteile

- Immer noch unzuverlässig und langsam
- Clients können sich der Super-Node-Aufgabe verweigern



FastTrack

- ▶ **Entwickelt von Niklas Zennström, Janus Friies, Jaan Tallinn 2001**
 - Autoren auch von Skype (P2P-Internet-Telefonie)
- ▶ **Hybride Peer-to-Peer-Netzwerk-Struktur**
 - mit Super-Nodes mit besonderen Aufgaben
 - Software entdeckt die Super-Node-Fähigkeit eines Peers
 - z.B. mehr Bandbreite, bessere Netzwerkverbindung
 - Super-Nodes für Lookup
 - Download geschieht über HTTP-Protokoll (direkt vom Client)
- ▶ **Software**
 - wurde nie veröffentlicht
 - der offizielle Client (Kazaa) enthält Malware
 - Client-Supernode-Kommunikation mittlerweile durch Reverse Engineering bekannt
 - Malware-freie Clients sind nun erhältlich (Kazaa lite)
 - Inter-Super-Node-Kommunikation noch unbekannt

E-Donkey

▶ Besteht aus einer Client-Server-Struktur

- Server
 - spezieller Server-Software
 - * z.B. Lugdunum
 - geben freiwillig Bandweite
 - speichern Index-Information und Inhalte
- Clients
 - verschiedene Client-Software
 - * z.B. eMule (am populärsten), Shareaza, MLDonkey, eDonkey2000, Hydranode, Morpheus, ...

▶ Clients laden die Dateien direkt von den Servers

▶ Diskussion:

- Anfällig für Attacken
 - Denial-of-Service, oder ähnliches (z.B. Razorback2-Server im Feb. 2006 beschlagnahmt von der Belgischen Polizei)
- Echtes Peer-to-Peer-Netzwerk?

Kademlia & Overnet

▶ **Kademlia ist eine Erweiterung von Edonkey**

- von Petar Maymounkov und David Mazzières
- ersetzt Server von Edonkey

▶ **Prinzip**

- jeder Client erhält eine ID durch Operation auf IP-Adresse und Kommunikationsfähigkeit
- Dateien werden ebenfalls durch IDs identifiziert
- Jeder Peer hat Kanten zu den k-nächsten Peers bezüglich einer Xor-Metrik
 - $\text{Distance}(A,B) = \text{Anzahl der 1er in String}(A \text{ Xor } B)$

- Index-Dateien werden auf den nächstens IDs gespeichert

▶ **Die Netzwerkstruktur von Kademlia orientiert sich an einem Hyperwürfel**

- Daher suche in logarithmischer Hop-Anzahl

▶ **Overnet**

- verwendet Kademlia-Protokoll
- ist eDonkey-Erweiterung

Juristische Situation

- ▶ **“IAAL*: What Peer-to-Peer Developers Need to Know about Copyright Law“, Fred von Lohmann, 2006**
 - Rechtliche Lage in den USA
- ▶ **Direct Infringement (Urheberrechtsverletzung)**
 - Zur Verfügungstellung von Musik, Dokumenten, Videos, etc. ist nur mit der Erlaubnis des Copyright-Besitzers zulässig
- ▶ **Secondary Infringement (Indirekte Urheberrechtsverletzung) durch P2P-Netzwerk-Betreiber/Entwickler**
 - Veranlassung
 - falls Urheberrechtsverletzung (auch von dritten) vorliegt und
 - diese vom Netzwerk-Entwickler/Betreiber unterstützt wird und
 - mit Vorsatz betrieben wird

- Beihilfe
 - falls Urheberrechtsverletzung (auch von dritten) vorliegt und
 - das Wissen darüber vorlag und
 - materiell dies unterstützt hat,
 - * z.B. durch Rechner, Sites, Speicherplatz, etc.
- Haftung für das Verhalten Dritter
 - falls Urheberrechtsverletzung (auch von dritten) vorliegt und
 - das Recht und die Fähigkeit zur Kontrolle vorlag und
 - ein finanzieller Nutzen daraus entstand.

Juristische Situation

- ▶ **“IAAL*: What Peer-to-Peer Developers Need to Know about Copyright Law“, Fred von Lohmann, 2006**
 - Rechtliche Lage in den USA
- ▶ **Empfehlungen:**
 - Keine Kopien machen oder Speichern
 - noch nicht einmal im RAM
 - Keine Werbung für Copyright-Verletzungen
- ▶ **Optionen**
 - Entweder totale Kontrolle
 - oder gar keine Kontrolle
- ▶ **Software**
 - Besser stand-alone statt Dienstleistung
- ▶ **Legalen Nutzen belegen**
- ▶ **Funktionen auslagern**
 - Beispiel Videorekorder und Videoband
- ▶ **Keinen finanziellen Nutzen aus Urheberschutzverletzungen ziehen**
- ▶ **Keine End-User-Licence-Vereinbarung**
- ▶ **Vorsicht beim Kunden-Support**
- ▶ **Open Source (!)**

Anwendungen von Peer-to-Peer-Netzwerken

- ▶ **File-Sharing ...**
- ▶ **Internet-Telefoniererei**
 - z.B. Skype
- ▶ **Verteilter Software-Update**
 - z.B. in P2P-Clients oder Spielen
- ▶ **Group-Ware**
 - ermöglicht störungsfreie Zusammenarbeit
- ▶ ...

Ausblick

- ▶ **Schwierigkeiten von Peer-to-Peer**
 - Client-Server-Architektur
 - die Eskalation der juristischen Situation (?)
 - asymmetrische Verbindungen
 - Firewalls, etc.
- ▶ **Diskussion**
 - Ist Peer-to-Peer die Netzwerkstruktur
 - der Demokratie oder
 - der Anarchie?
 - Rechtliche Situation



Systeme II

Ende der 12. Vorlesungswoche

Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Rechnernetze und Telematik
Christian Schindelhauer
Sommer 2008