HUMBOLDT-UNIVERSITÄT ZU BERLIN

Peer-to-Peer-Systeme

Teil II: Filesharing in unstrukturierten Overlays

Björn Scheuermann

Humboldt-Universität zu Berlin
Wintersemester 2015/16

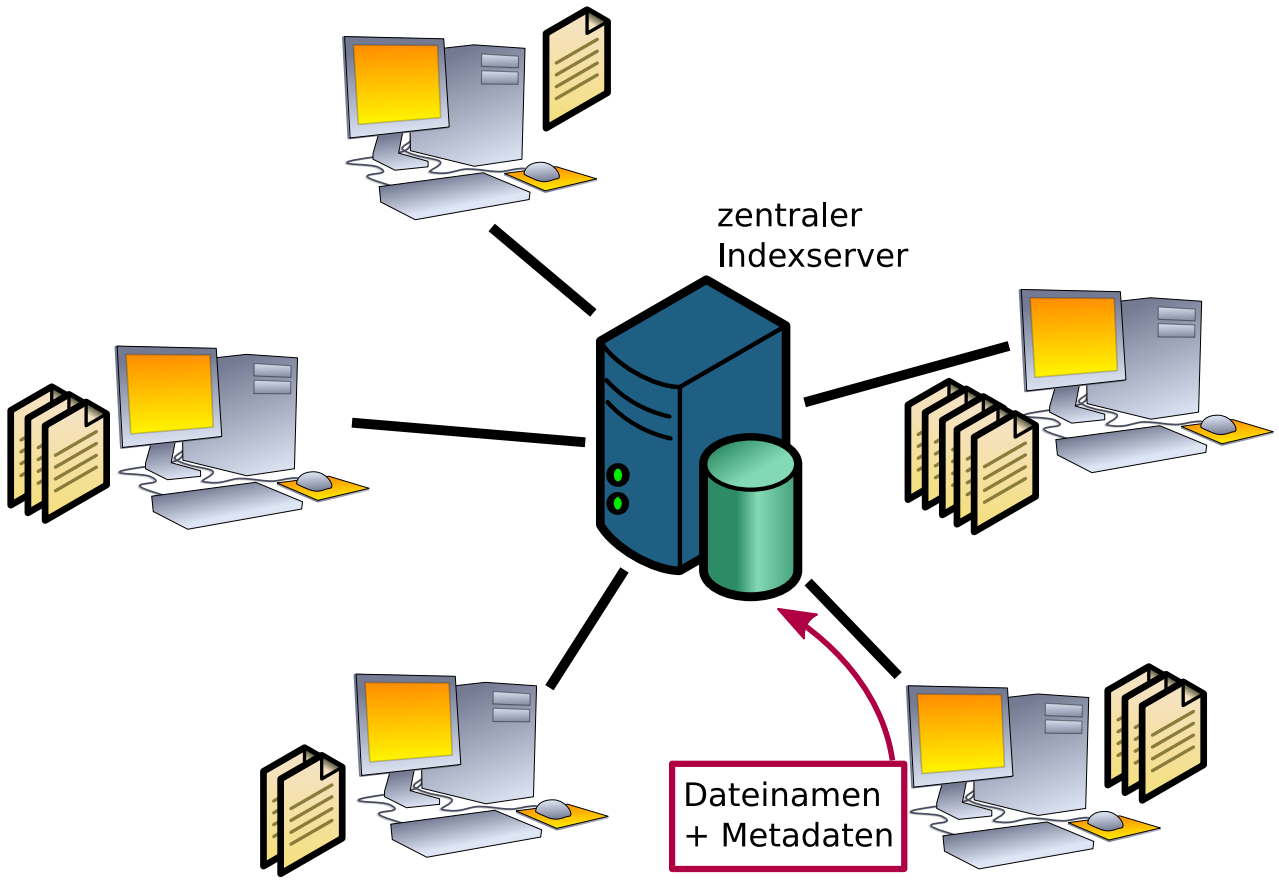
Napster

- Das erste Filesharing-System
- Im Juni 1999 von Shawn „Napster“ Fanning veröffentlicht
- Zum Tauschen von MP3-Dateien verwendet
- Erste Klage der RIAA im Dezember 1999, weitere folgten
- Das ursprüngliche Napster ging 2001 wieder vom Netz
- Jetzt: Kommerzielle Plattform unter demselben Namen

[Röttgers: Mix, Burn & R.I.P. – Das Ende der Musikindustrie, Heise-Verlag, 2003]

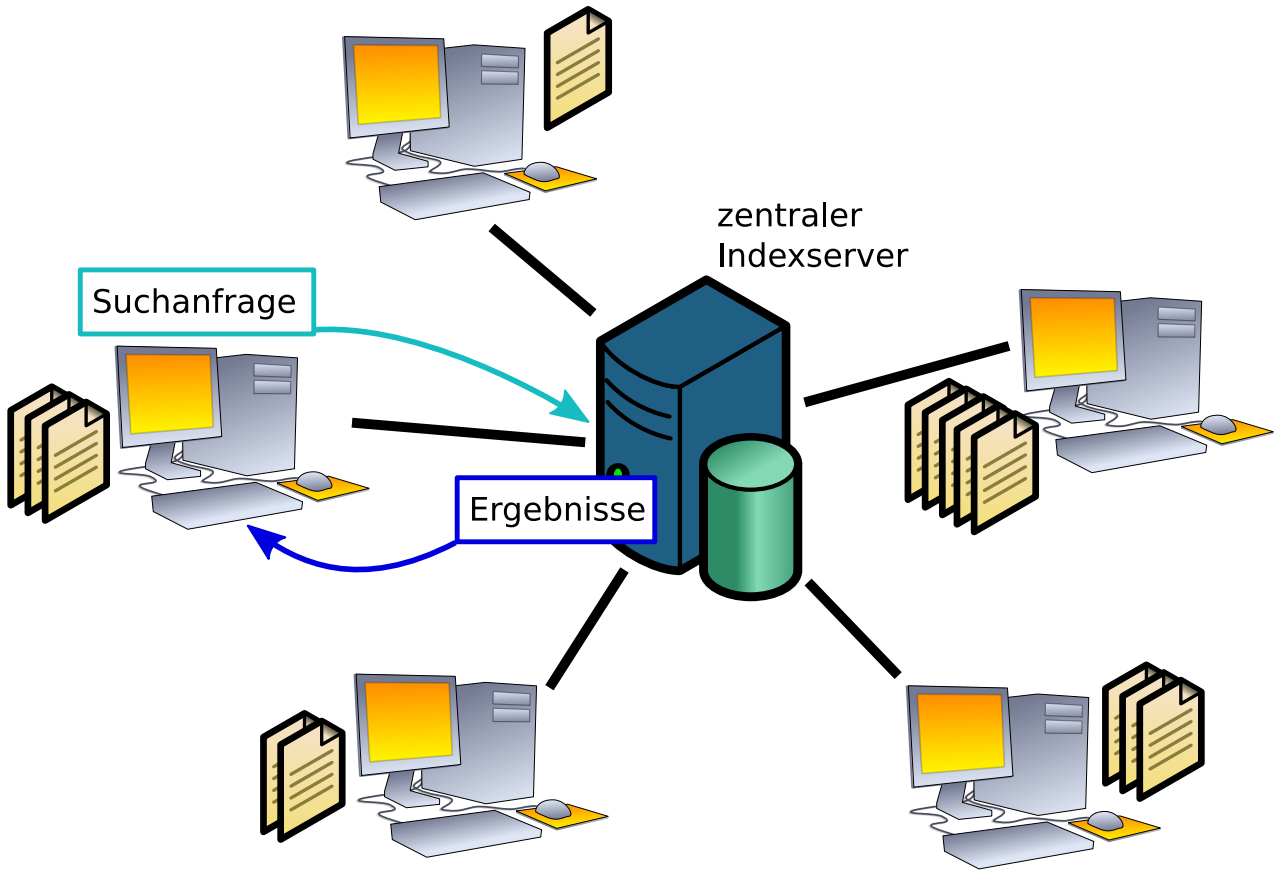
Funktionsweise von Napster

- Alle Clients sind mit einem zentralen Server verbunden
- Der Server pflegt einen zentralen Index bereitgestellter Dateien



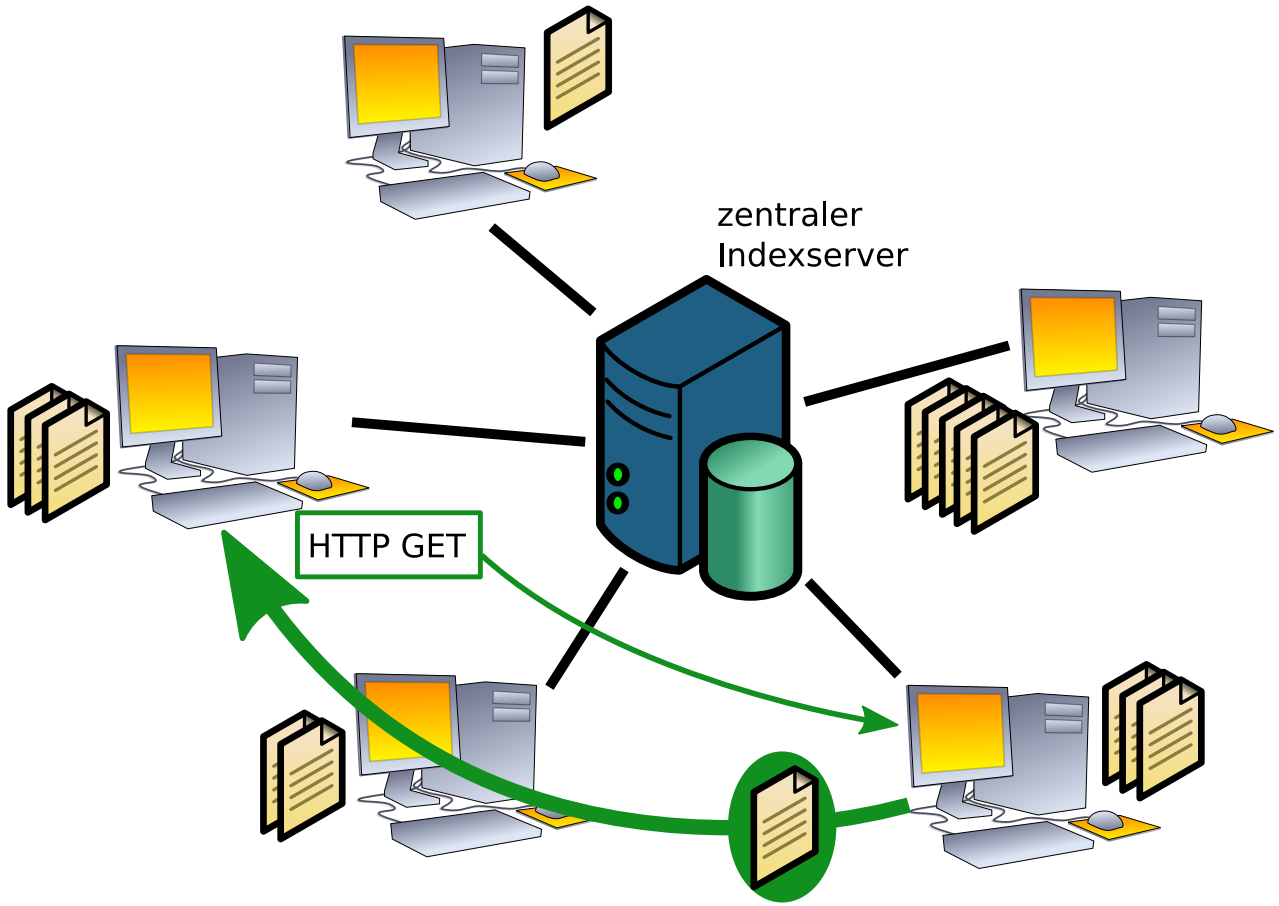
Funktionsweise von Napster

- Zum Suchen einer Datei stellt der Client eine entsprechende Anfrage an den Server
- Der Server antwortet mit einer Trefferliste
- Enthält IP-Adressen von Clients, die die Datei anbieten



Funktionsweise von Napster

- Die eigentliche Dateiübertragung erfolgt ohne Beteiligung des Servers
- Signalisierung über HTTP



Napster – Diskussion

- Einfach!
- Alle Dateien können in einem Schritt gefunden werden (Nachrichtenkomplexität $O(1)$)
- Ohne den zentralen Index kann keine Datei gefunden werden

$\Rightarrow$ Zentraler Server ist ein Single-Point-of-Failure und macht das System angreifbar

- Rechenleistung, Speicher und Bandbreite des zentralen Servers müssen mit der Zahl der Benutzer wachsen
- Eigentlich gar kein „richtiges“ Peer-to-Peer-Netzwerk!

Gnutella

- Gnutella wurde im März 2000 von Justin Frankel und Tom Pepper (beide Nullsoft) vorgestellt
- Das erste vollständig dezentrale Filesharing-System
- Die Peers in Gnutella heißen „Servents“
- Zwei Protokollversionen:
  - Gnutella 0.4
  - Gnutella 0.6

Bootstrapping-Problem

- Um Kontakt zu anderen Peers aufzubauen, muss man die Adresse mindestens eines Netzwerkteilnehmers kennen
- Bei Gnutella gibt es keine festen, zentralen Netzwerkteilnehmer

Wie tritt man einem vollständig dezentralen Netzwerk bei?

Bootstrapping in Gnutella 0.4

- Mit der Software mitgeliefert wird eine Liste von „stabilen“ Peers
- Beim ersten Aufruf wird diese Liste durchprobiert, bis ein aktiver Peer gefunden ist
- Von diesem Peer aus wird das Netzwerk erkundet
- Dadurch wird die Liste bekannter aktiver Peers („Host Cache“) erweitert und aktualisiert
- Für den nächsten Aufruf wird der Host Cache persistent gespeichert

GWebCache

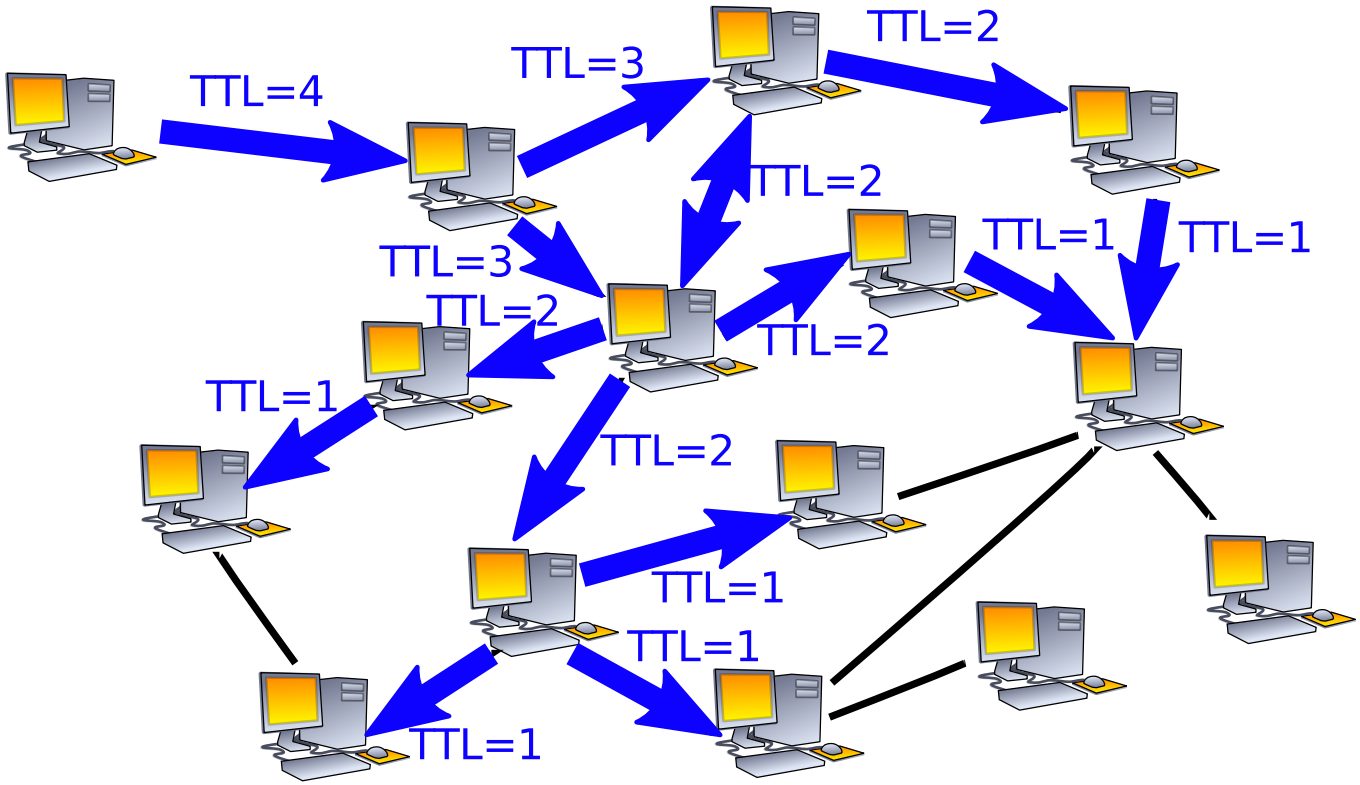
- Neuere Servents können alternativ bzw. zusätzlich Webserver benutzen, um Adressen bekannter Peers auszutauschen
- GWebCache-Server liefern auf Anfrage Listen von
  - aktiven Gnutella-Peers
  - anderen GWebCache-Servern
- Servents registrieren sich nach einer gewissen Aktivitätsdauer bei einem (oder mehreren) GWebCaches

Gnutella – Ping

- Das Erkunden des Netzwerks erfolgt mittels Ping-Nachrichten
- Ping-Nachrichten enthalten drei wesentliche Felder:
  - Message-ID als eindeutige Kennzeichnung (16-Byte-String)
  - Time-to-Live-Feld (TTL, ähnlich wie in IP)
  - Hop-Count-Feld („inverse“ TTL)
- Ping wird im Netzwerk geflutet
  - ein empfangender Knoten leitet ein Paket an alle seine Nachbarn weiter (außer an den, von dem er selbst das Paket erhalten hat)
  - außerdem merkt er sich die Message-ID des Pings und den Nachbarn, von dem er ihn erhalten hat
  - später von anderen Nachbarn eintreffende Duplikate werden verworfen (Vergleich der Message-ID)

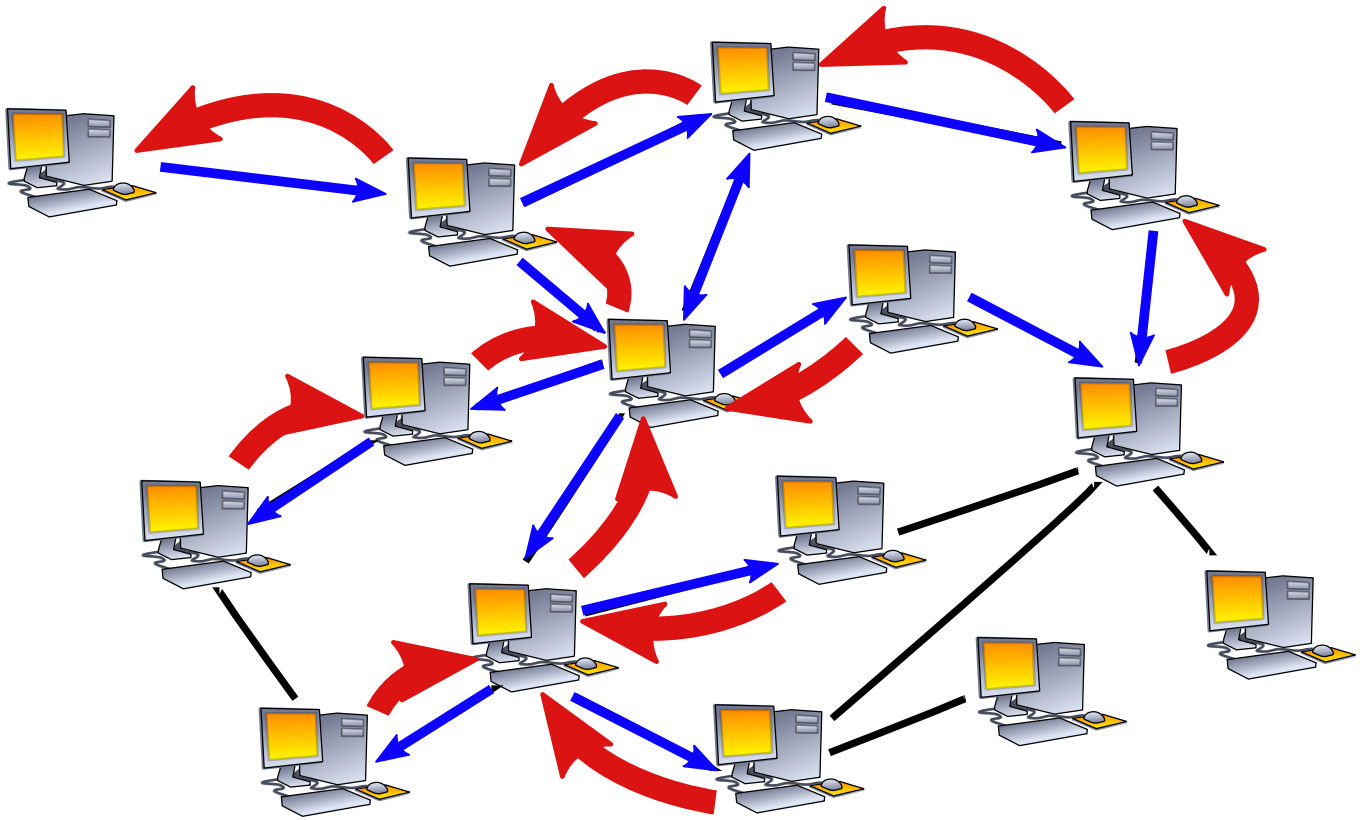
Gnutella – Ping

- Tiefenbeschränkung durch ein Time-to-Live-Feld (TTL)
  - TTL wird bei jedem Schritt um 1 verringert
  - wenn TTL $= 0$, wird die Nachricht verworfen



Gnutella – Pong

- Knoten antworten auf einen Ping mit einer Pong-Nachricht
  - enthält IP-Adresse + Port des Peers, ID des Ping
  - wird auf dem umgekehrten Weg zurückübertragen (Reverse-Path-Routing)
  - dafür werden die gemerkten Daten über den Weg des Ping verwendet

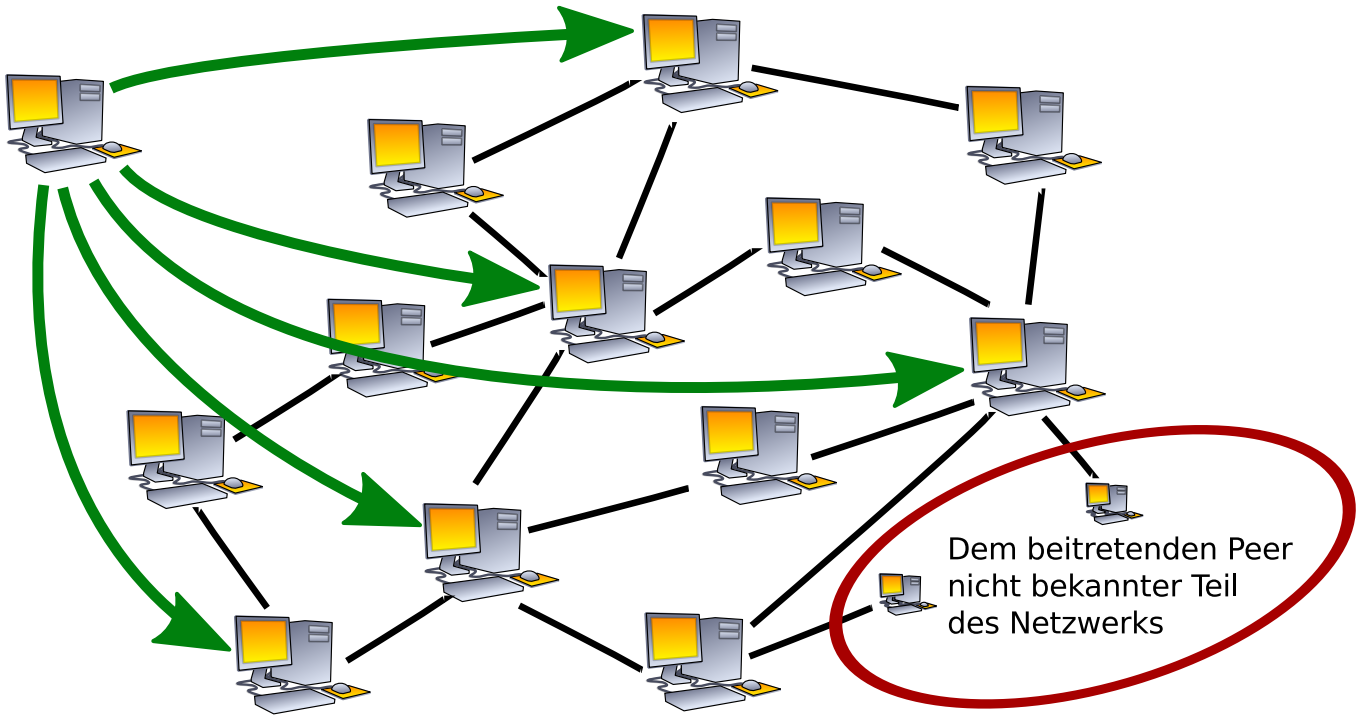


Pong-Caching

- Ping+Pong erzeugen sehr viel Datenverkehr
- Neuere Protokollversionen verwenden deshalb „Pong-Caching“:
  - Knoten leiten eingehende Pings nicht bzw. nicht unmittelbar im Overlay weiter
  - Pings werden statt dessen mit einer Auswahl (typ. ca. 10) zuvor gespeicherter Pongs beantwortet
  - gespeicherte Pongs werden über eigene periodische Pings aktuell gehalten
- Viele Varianten, hohe Komplexität im Detail!

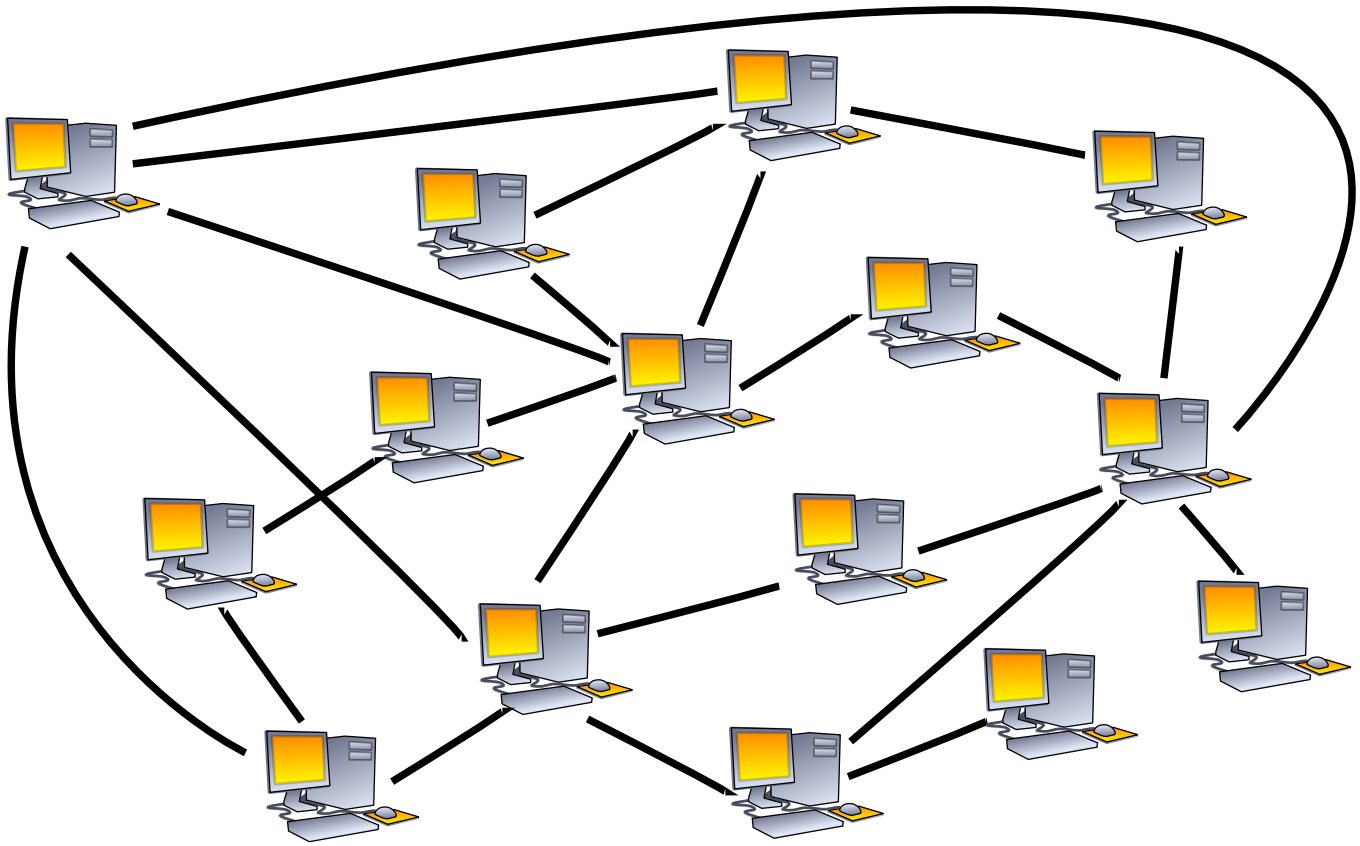
Gnutella – Join

- Nach Eintreffen der Pongs sind viele aktive Peers bekannt
- Der neue Peer wählt nun n davon zufällig aus (typ. $n = 5$) und baut TCP-Verbindungen zu ihnen auf



Gnutella – Join

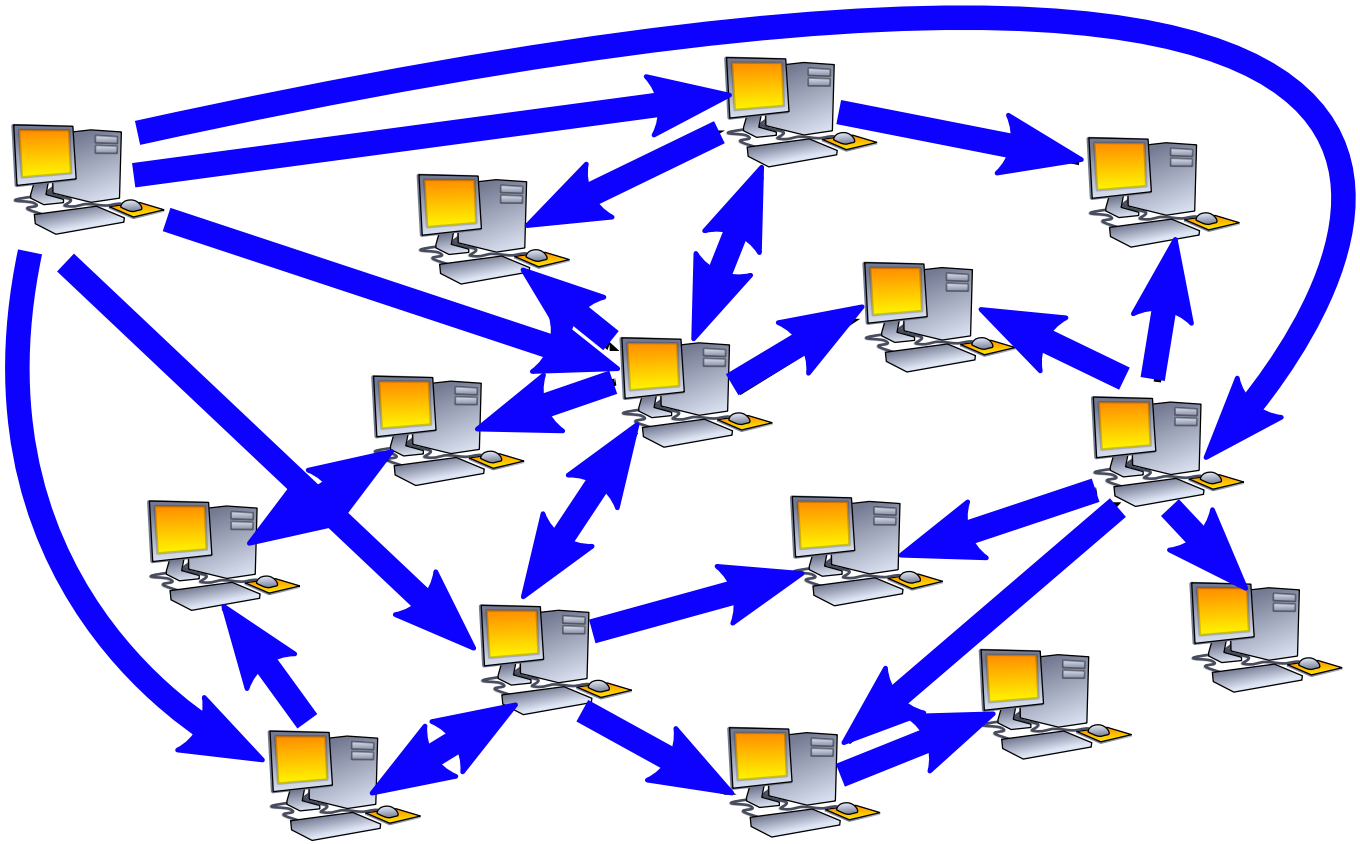
- Der Peer ist nun Teil des Gnutella-Netzwerks



⇒ keine zentralen Kontrollmechanismen,
ein zufallsgesteuerter Prozess erzeugt das Overlay

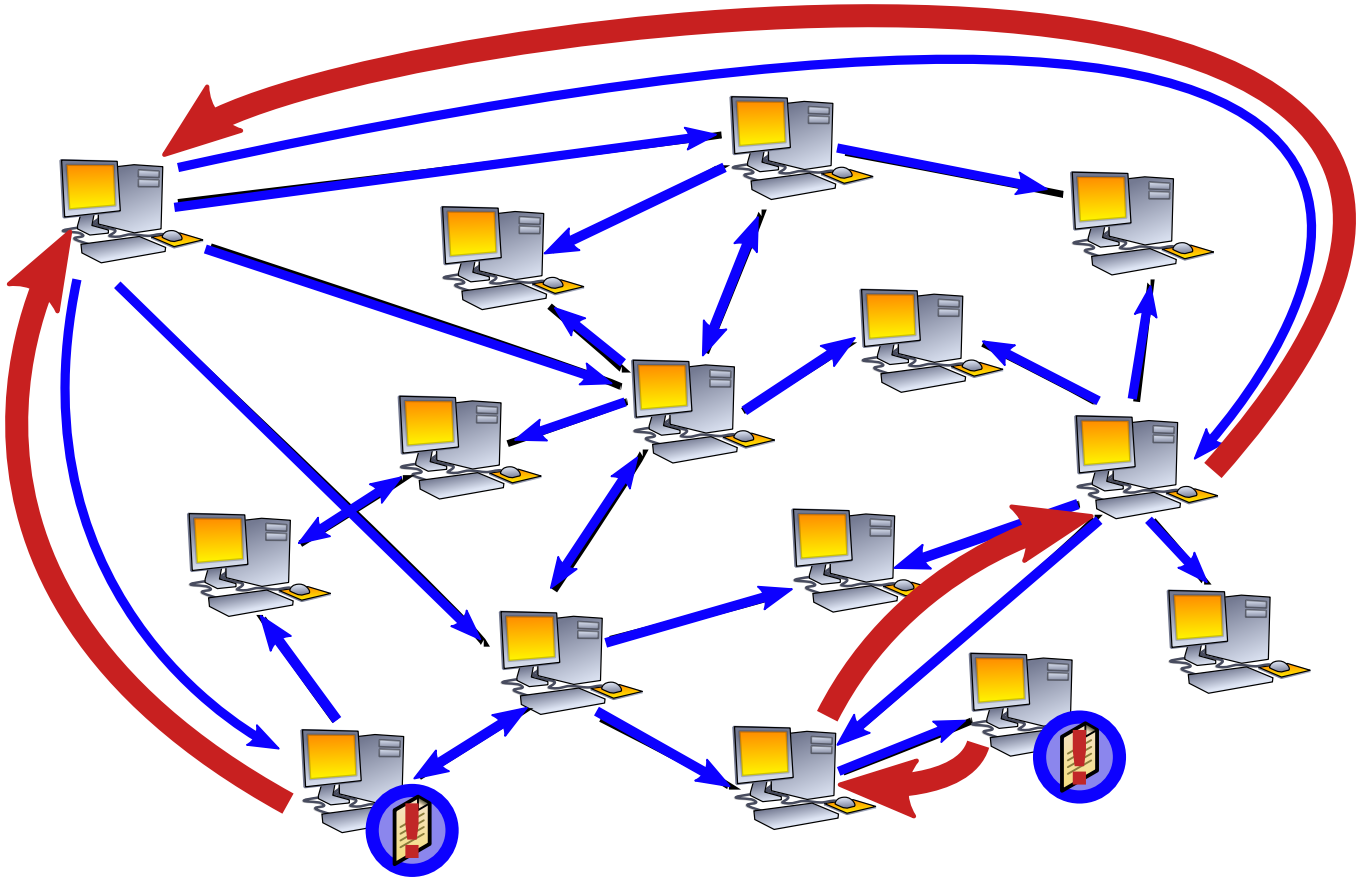
Gnutella – Suche

- Query-Nachrichten werden für die Suche nach Dateien verwendet
- Sie werden wie Ping-Nachrichten über k Hops geflutet (typ. $k = 7$)



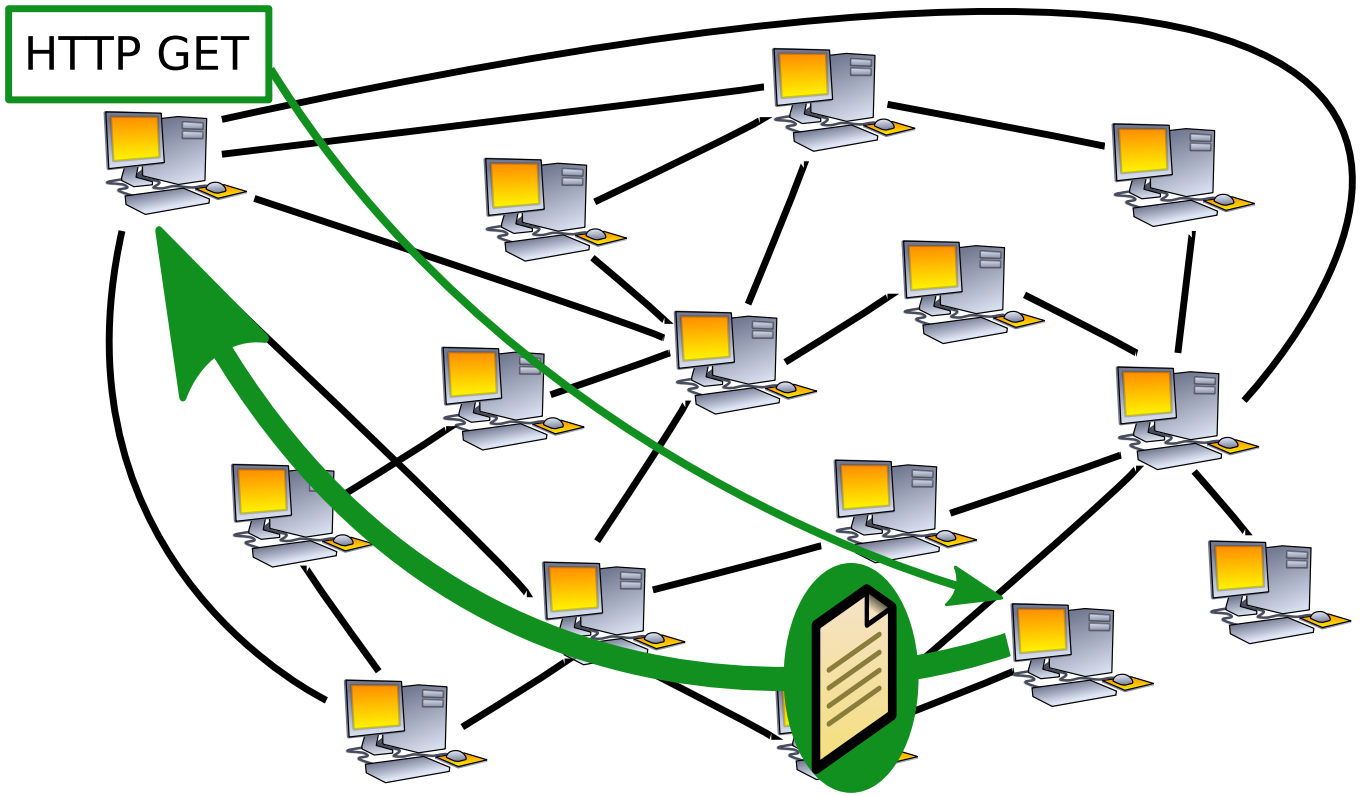
Gnutella – Suche

- Treffer werden über QueryHit-Nachrichten signalisiert
- Im Overlay weitergeleitet wie Pong-Nachrichten (Reverse-Path-Routing)



Gnutella – Download

- Das Herunterladen erfolgt wie bei Napster direkt zwischen den Peers
- Auch hier erfolgt die Signalisierung per HTTP



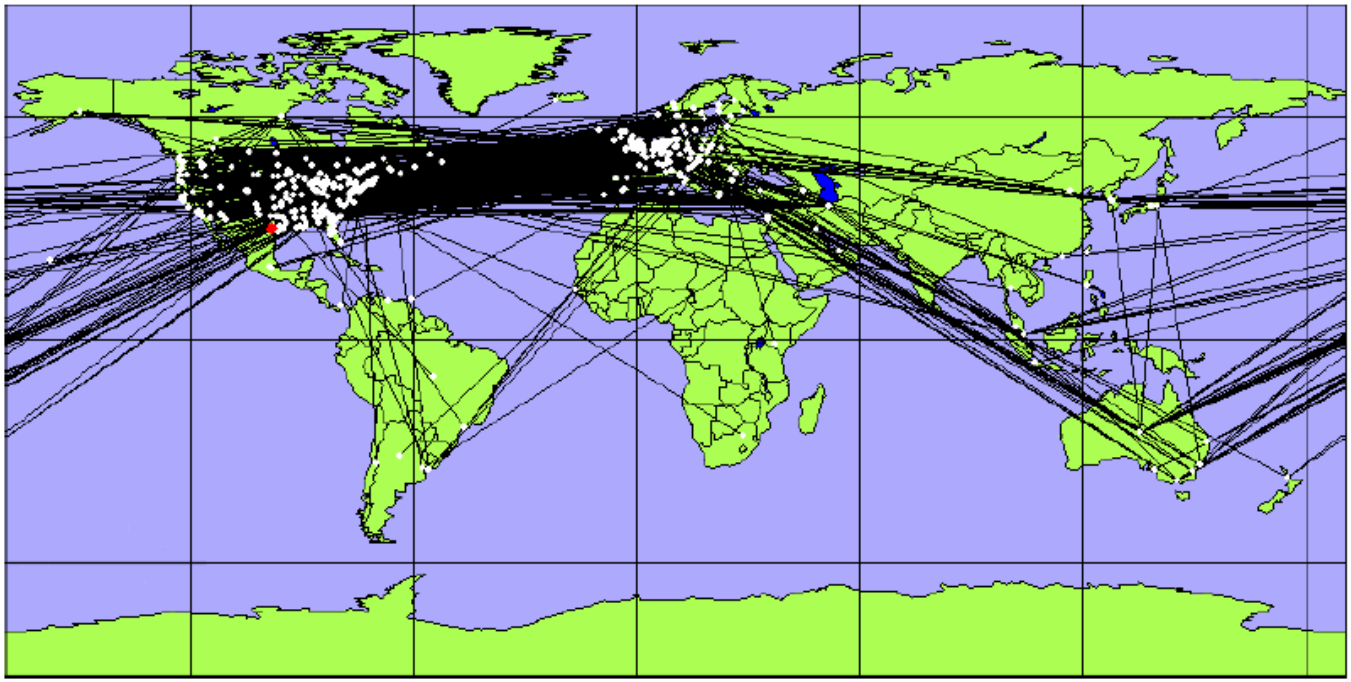
TTL vs. Hop-Count

Gnutella-Nachrichten haben sowohl einen (aufwärts zählenden) Hop-Count-Zähler als auch ein (abwärts zählendes) TTL-Feld. Warum?

- Suchanfragen mit hoher TTL sind vorteilhaft für den Anfrager, aber mit sehr viel Aufwand für das Netzwerk verbunden
- Kombination von TTL + Hop-Count ermöglicht es, dass jeder weiterleitende Knoten (nicht nur direkte Nachbarn) Anfragen mit hohen anfänglichen TTLs erkennen kann

Zickzack-Routen

- Gnutella orientiert sich nicht am darunterliegenden Netzwerk
- Das führt zu „Zickzack-Routen“
- Ausschnitt aus dem Gnutella-Netzwerk am 1.8.02:



(Abbildung: [Schollmeier, Kunzmann: GnuViz – Mapping the Gnutella Network to its Geographical Locations, PIK 2003])

⇒ lange Verzögerungen, viel Verkehr auf „teuren“ Links (Inter-AS, interkontinental,. . .)

Gnutella – Zwischenfazit

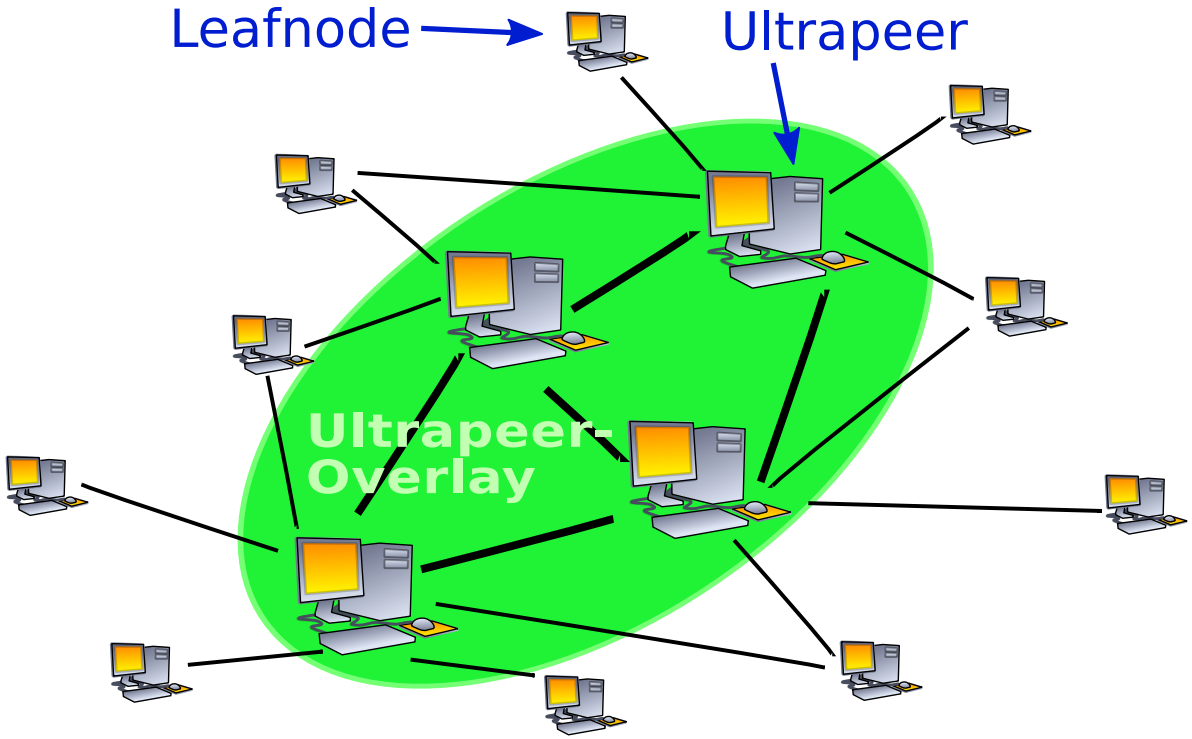
- Erstes „echtes“ Peer-to-Peer-Netzwerk
- Extrem robust
- Aber:
  - lange Antwortzeiten wegen des exzessiven Flutens
  - durch tiefenbeschränkte Suche wird nur in einem Teilnetzwerk gesucht
  - seltene Daten werden nur mit Glück gefunden
  - könnte man durch höhere TTL vermeiden, aber dann steigt der Nachrichtenaufwand stark an

Hierarchische Peer-to-Peer-Netzwerke

- Zentralisierte Systeme wie Napster können sehr viel effizienter suchen
- Idee: Hierarchische Peer-to-Peer-Netzwerke
- „Zwischending“ zwischen zentralisierten und verteilten Systemen
- Zwei Sorten von Peers:
  - Super-/Ultrapeers
  - Leafnodes
- Ursprünglich eine Idee aus KaZaA/FastTrack, umgesetzt dann auch in Gnutella 0.6

Gnutella 0.6 – Overlay-Struktur

- Ultrapeers bilden untereinander ein „echtes“ Peer-to-Peer-Netzwerk
- Leafnodes sind mit einem oder mehreren Ultrapeers (typ. heute: 1–3) verbunden
- Ultrapeers bedienen typ. bis zu 30–45 Leafnodes



Query Routing Protocol (QRP)

- Leafnodes registrieren ihren Dateiindex bei ihrem Ultrapeer (im Prinzip eine Liste von Keywords, bei Gnutella „Routing-Tabelle“ genannt)
- Suche wird zunächst nur im Ultrapeer-Overlay verteilt
- Ultrapeers geben die Suchnachricht an die Leafnodes weiter, deren Routing-Tabelle passende Einträge hat („Query Routing“)
- Andere Leafnodes werden abgeschirmt

$\Rightarrow$ Effizientere Suche, da sehr viel weniger Peers die Suche bearbeiten müssen

Dynamic Querying

- Idee: Passe die Suche so an, dass Suchanfragen mit vielen Treffern nicht so weit verbreitet werden
- Ultrapeer geht dafür in mehreren Schritten vor:
  1. Suche zunächst bei eigenen Leafnodes
  2. Wenn das nicht reicht (typ.: 150 Treffer = genug), suche mit kleiner TTL und ggf. nur bei manchen Nachbarn
  3. Wenn das immer noch nicht reicht, bestimmt die TTL und Nachbaranzahl, die wahrscheinlich eine ausreichende Anzahl an Treffern liefern wird; verwende die Ergebnisse von Schritt (2), um diese Werte abzuschätzen

Join im hierarchischen Gnutella

- Bootstrapping verwendet GWebCache und Pong-Caching (durch Ultrapeers)

- Wenn ein neuer Leafnode beitreten will und einen aktiven Servent kontaktiert, sind zwei Fälle möglich:
  - kontaktierter Peer ist Ultrapeer $\Rightarrow$ super :-)
  - kontaktierter Peer ist Leafnode $\Rightarrow$ „umleiten“ an den Ultrapeer

Wie wird ein Servent zum Ultrapeer?

- Empfehlungen in Gnutella für potentielle Ultrapeers:
  - keine (NAT-)Firewall
  - Betriebssystem, das mit vielen offenen Sockets gut klarkommt
  - hohe Bandbreite der Internetanbindung
  - hohe Uptime
  - ausreichend CPU-Leistung und Speicher

Wie wird ein Servent zum Ultrapeer?

- Ein „Ultrapeer-fähiger“ Servent signalisiert dies beim Handshake
- Er wird als Ultrapeer oder Leafnode eingebunden – je nachdem, ob Ultrapeers benötigt werden
- So passt sich das Netzwerk selbst an die aktuellen Erfordernisse an
- Prinzipielles Problem: Warum sollte ein Knoten eigentlich Ultrapeer werden wollen?

Modernes Gnutella – Diskussion

- Hierarchisches Overlay reduziert den Nachrichtenaufwand stark
- Hierarchie und weitere Mechanismen (GWebCache, Pong-Caching, Dynamic Querying,. . .) sorgen für bessere Skalierbarkeit und höhere Effizienz in neueren Gnutella-Varianten
- Aber auch sehr viel zusätzliche Komplexität!

Zusammenfassung

- Wir haben die frühen Filesharing-Systeme Napster und Gnutella kennen gelernt
- Während Napster einen zentralen Indexserver verwendet, ist Gnutella das erste vollständig dezentrale System
- Wir haben gesehen, wie ein vollständig verteiltes Overlay ohne zentrale Koordination aufgebaut wird und wie darin gesucht werden kann
- Schließlich haben wir Weiterentwicklungen des ursprünglichen Gnutella-Netzes betrachtet und gesehen, wie ein hierarchiches Overlay die Skalierbarkeit verbessern kann