Seminar Moderne Konzepte für weitverteilte Systeme SS 02

Filesharing mit Gnutella: Skalierungsprobleme eines populären P2P-Systems

Torsten Ehlers

10.06.2002

Übersicht

- Gnutella: Eigenschaften des Systems
- Das Gnutella-Protokoll
- Skalierungsprobleme
- Mögliche Lösungsansätze
- Flow Control Algorithm

Gnutella

- verteiltes P2P Filesharing-System
- dezentral
- Teilnehmer sind Clients und Server, *Servents*
- keine Hierarchie
- teilweise zyklische Netzstruktur

Gnutella (2)

- 1. Implementierung durch Nullsoft am 14.03.2000 („Gnutella v0.56“)
- zahlreiche weitere Clients (Bearshare, LimeWire, Morpheus, ...)

Gnutella (3)

- kein *single point of failure*
- schwierig angreifbar, da kein zentraler Server
- aber: DoS-Angriffe möglich

- hohe Fluktuation der Teilnehmer und damit auch Daten

Das Gnutella-Protokoll

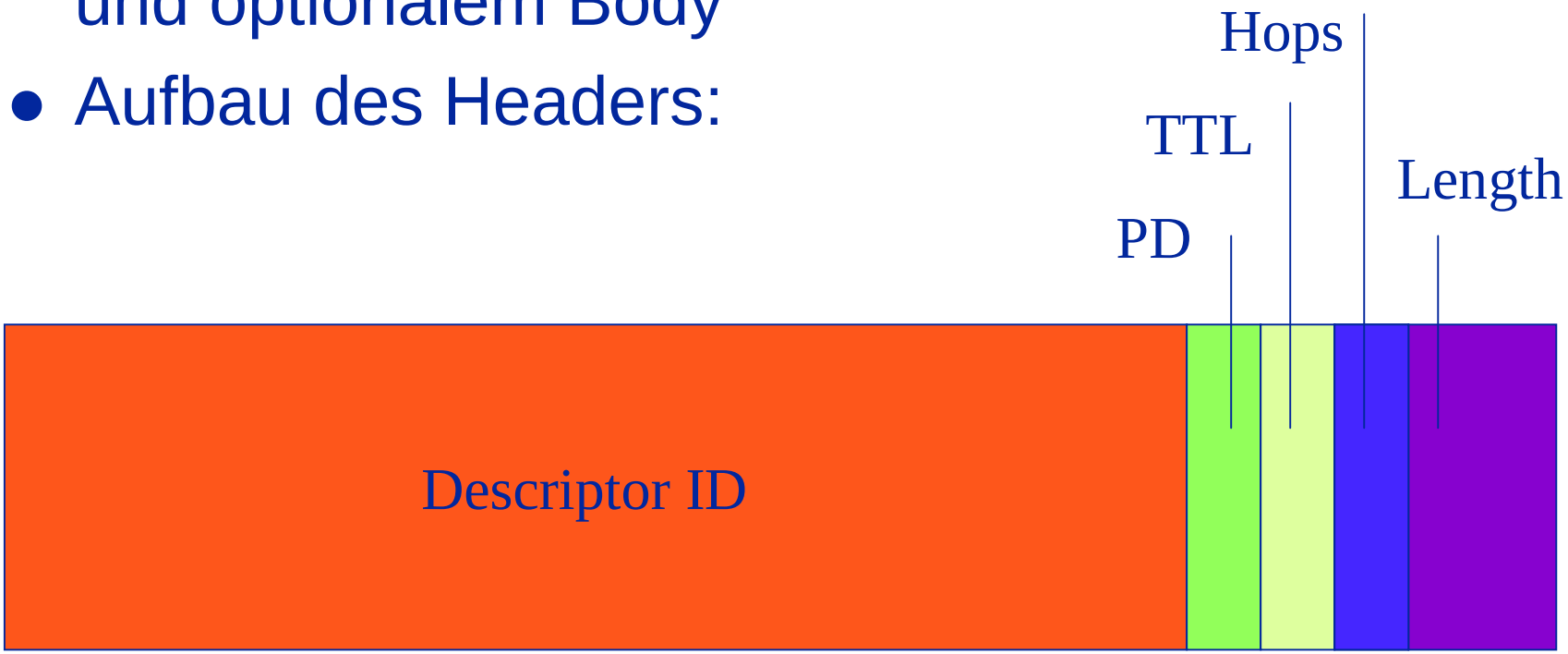
- Regelt die Kommunikation der Clients untereinander
- nutzt TCP
- Aufbau von Servent-Verbindungen über CONNECT / OK Nachrichten
- i.d.R. bis zu 4 Verbindungen

Das Gnutella-Protokoll (2)

- Kommunikation ausschließlich über spezielle Pakete, *Deskriptoren*
- Netz dient der Suche nach Dateien
- Keine Übertragung von Nutzdaten im Gnutella-Netz

Deskriptoren

- Deskriptor besteht aus 22 Bytes Header und optionalem Body
- Aufbau des Headers:



Deskriptoren (2)

- ID: Eindeutige ID (16 Bytes)
- Payload Descriptor: Ping, Pong, Query, Query Hit oder Push (1 Byte)
- TTL = Time To Live (1 Byte)
- Hops = Zahl der erfolgten Weiterleitungen (1 Byte)
- Payload Length = Länge der Payload in Bytes (3 Bytes)

Versand von Deskriptoren

- Bei jedem Weiterleiten Hop-Feld inkrementieren
- Bei Hops = TTL Deskriptor verwerfen
- identische Deskriptoren (ID und Payload-Deskriptor gleich) nicht erneut versenden

Ping - Deskriptor

- Zweck: Durchsuchen des Netzwerks nach weiteren Servents
- wird z.B. über neu aufgebaute Verbindung versandt
- wird als Broadcast über alle anderen Verbindungen weitergeleitet
- kein Body
- wird mit Pong-Deskriptor beantwortet

Pong - Deskriptor

- Antwort auf einen Ping-Deskriptor
- wird zurückgeroutet
  - Routing Tabelle mit Descriptor ID und Verbindung
  - zurückschicken an Vorgänger
  - IDs stimmen überein
- Body enthält IP, Port, Zahl und Gesamtgröße der Dateien
- Empfänger kann Verbindung dorthin öffnen

Query - Deskriptor

- Dient der Dateisuche
- enthält Suchstring und Mindestgeschwindigkeit im Body
- Weiterleitung per Broadcast (analog zu Ping)
- Anonymität beim Suchen
- Wird ggf. mit Query-Hit-Deskriptor beantwortet

Query – Hit – Deskriptor

- Antwort auf einen Query-Deskriptor, falls gesuchte Datei vorhanden
- wird analog Pong-Deskriptor zurückgeroutet
- Body enthält IP, Port, Geschwindigkeit, Ergebnismenge

Push - Deskriptor

- wird für Hosts mit privater IP benötigt (Aufforderung zur Verbindungsherstellung)
- hier nicht betrachtet

Übertragen der Nutzdaten

- Kein Datenverkehr im Gnutella-Netz
- Direktverbindung (HTTP)
- Aufgabe der Anonymität

Skalierungsprobleme

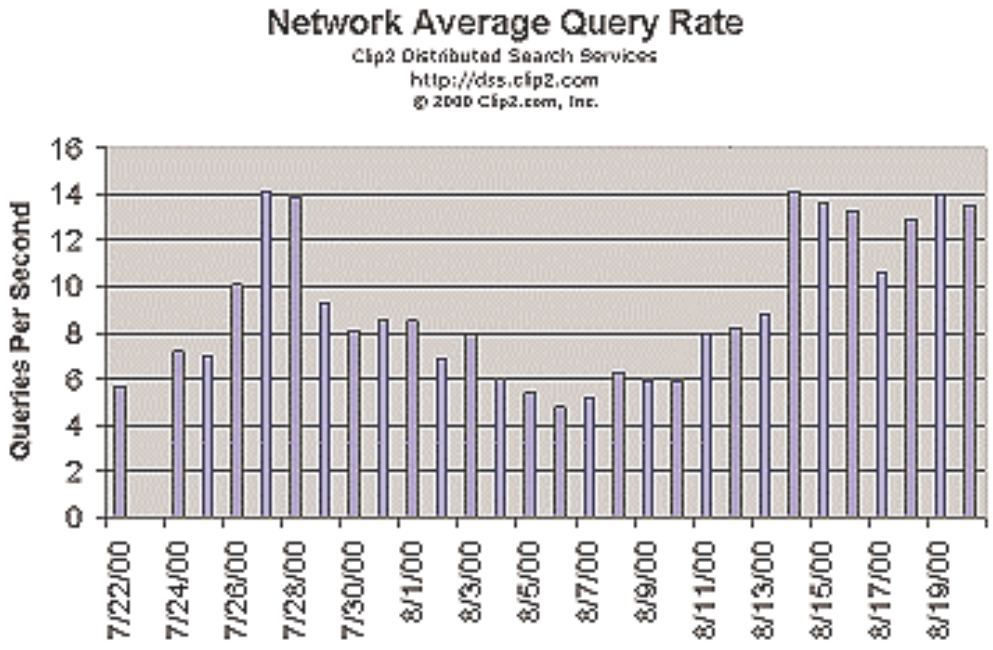
- Ziel: Netz toleriert steigende Nutzerzahlen
- mehr Nutzer bedeuten mehr Anfragen
- dezentral, jeder muß Serverfunktion übernehmen
- Dateiverteilung unstrukturiert
- Suchanfragen nach Schneeballprinzip aufwendig (Vervielfachung mit jedem Hop)

Skalierungsprobleme (2)

- überflüssige Nachrichten durch zyklische Struktur
- zu hoher Traffic überfordert zunächst Teilnehmer mit schwächster Bandbreite

Beispiel

- Network Average Query Rate (NAQR): Durchschnittliche Zahl der Anfragen pro Host (eigene und weitergeleitete) pro Sekunde



Beispiel (2)

- Durchschnittlich
  - 3 Verbindungen pro Host
  - 30 Bytes pro Nachricht (70 mit Header der unteren Schichten)
  - NAQR = 10
  - Anteil der Queries am Datenaufkommen = ¼
- 3x70x10x4 = 8400 Bytes/s oder 67,2 KBit/s (gegenüber 56 KBit/s Modem)

Skalierungsprobleme (3)

- Fallen mehr Daten an als Host senden kann, kann er die Daten verwerfen oder puffern
- a) verwerfen
  - willkürliches Verwerfen ungünstig
  - Query Hits: bereits großer Traffic durch Query erzeugt

Skalierungsprobleme (4)

- b) puffern
  - sinnvoll bei kurzzeitigen Trafficbursts
  - bei dauerhaft zu großen Traffic Verschlimmerung des Problems
    - längere Antwortzeit
    - schließlich Pufferüberlauf

Skalierungsprobleme (5)

- auf Dauer scheidet überlasteter Host praktisch aus Netzwerk aus
- kann Netz in 2 Teile trennen
- betrifft dann auch Hosts mit Breitbandanbindung
- TTL hilft nur bedingt, v.a. keine Dynamik

Skalierungsprobleme (6)

- Netz nur funktionsfähig so lange Zahl der Servents beschränkt
- längere Antwortzeit
- Überlauf von IP-Tabellen
- geringerer Suchraum

Reliable Connection Problem

- Grundsätzliches Problem der Zuverlässigkeit
- Gnutella nutzt TCP
- TCP puffert (setzt auf kurzfristig zu große Datenmengen)
- keine Information der Anwendung, keine Reaktionsmöglichkeit
- TCP setzt auf Zuverlässigkeit um jeden Preis
- UDP keine Alternative

Lösungsansätze

- Unstrukturiertes Design soll beibehalten werden, da
  - gute Anpassung an hohe Fluktuation
  - Struktur bringt wenig bei „weiten“ Suchanfragen
- Vorschlag: Aufgabe des Schneeballprinzips, stattdessen nur einfache Weitergabe (gewichtet)
- Schneller aber nicht abwärtskompatibel

Eigenschaften einer Lösung

- Zahl der Host prinzipiell beliebig erhöhbar
- Berücksichtigung der Bandbreite einzelner Hosts
- Faire Verteilung der Bandbreite, Behinderung von DoS-Angriffen
- Abwärtskompatibilität

Flow-control Algorithmus

- Outgoing flow control block
- Q-algorithm block
- Fairness block

Outgoing flow control block

- steuert ausgehenden Datenfluß einer einzelnen Verbindung
- Ziel: Minimierung der Verzögerung über diese Verbindung
- Ausschalten der negativen Effekte der TCP-Puffer
- Eigenes Bestätigungsschema auf höherer Ebene ohne Puffern

Outgoing flow control block (2)

- Einfügen einer PING-Nachricht alle 512 Bytes (TTL 1)
- Keine Weiterleitung, nur Antwort mit PONG
- Neue Daten erst nach Erhalt des PONG senden
- Nie mehr als 512 Bytes + PING + PONG zwischen 2 Servents unterwegs

Outgoing flow control block (3)

- PING = 22 Bytes, PONG = 37 Bytes
- Overhead: 59 / (512 + 59) Bytes = 10,3 %

- Zusätzliche Daten werden nicht gepuffert, sondern verworfen
- Priorisierung der Daten im 512-Byte Fenster

Outgoing flow control block (4)

- Responses vor Requests
- Priorisierung innerhalb der Requests
  - niedriger Hop Count hat Vorrang
  - geringerer Anteil des Suchraums geht verloren
  - dynamische „lokale TTL“

Q-algorithm block

- befindet sich logisch auf Empfangsseite einer Verbindung
- entscheidet, welche Requests per Broadcast weitergereicht werden sollen
- Ziel: Nur so viele Requests versenden, daß Verbindung auch fällige Responses verarbeiten kann
- Besser Requests gar nicht erst zu senden, statt Responses verwerfen zu müssen

Q-algorithm block (2)

- Vorhersage der Responses sehr schwierig
- Menge und Verzögerung der Antworten vorher nicht bekannt
- Beeinflussung durch Outgoing Flow-Control Block
- Nur statistische Analyse basierend auf Vergangenheit möglich

Q-algorithm block (3)

- Möglichkeit einzelner überlastender Trafficburst bleibt bestehen, Wahrscheinlichkeit wird reduziert
- Nur Hälfte der Bandbreite einplanen (ohnehin Leitungsnutzung auch durch andere Applikationen)
- Pakete verwerfen erst ab Traffic, der Schnitt um 200% übersteigt, und dann v.a. Requests

Fairness block

- Erweiterung des Outgoing flow control blocks
- Faire Verteilung der Bandbreite für Suchanfragen auf einzelne Verbindungen
- Priorisierung nach Hop Count bereits durch Block 1
- Zweite Priorisierungsstufe nach Ursprungsverbindung einführen

Fairness block (2)

- Schlechte Lösungen
  - Alle Anfragen von Verbindung 1, dann 2, ... => Aushungern der hohen Nummern
  - Zufällig Anfragen auswählen => DoS-Angriff mit sehr vielen Anfragen => Aushungern der anderen
- Stattdessen: Round-Robin Verteilung
- Queue für jede Verbindung, sortiert nach Hop Count
- Abwechselnd Queues bedienen

Zusammenfassung

- ursprüngliches Gnutella-Protokoll relativ einfach
- ohne zusätzliche Vorkehrungen Gnutella schlecht skalierbar
- verschiedene Ansätze denkbar
  - Struktur
  - kein Broadcast
  - Priorisierung der Nachrichten