

# Peer-to-Peer Computing *Backstage*

Stefan Schmid

# In This Lecture...

---

What are the **dead sea scrolls** of peer-to-peer?

What has my **Playstation 3** to do with this summer school?

Why did Gnutella crash after the **inrush of former Napster users**?

How does BitTorrent **foster cooperation** among peers?

How can I use BitTorrent without **going to jail**?

How does a BitTorrent download differ from a HTTP download?

What is an **end-game**?

How to remove **Simpsons** from *Kad*?

What does Skype do when I am **not on the phone**?

Peer-to-peer **botnets**?



# Before We Go Backstage...

---

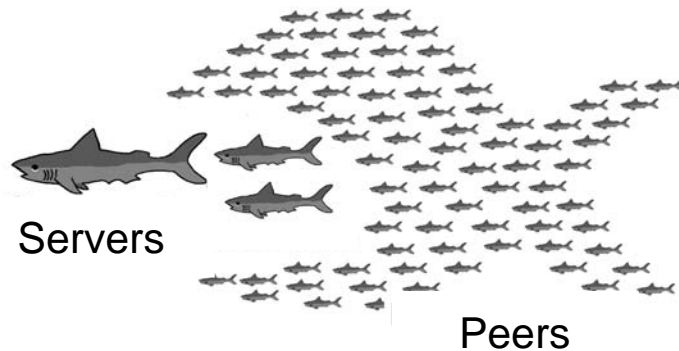
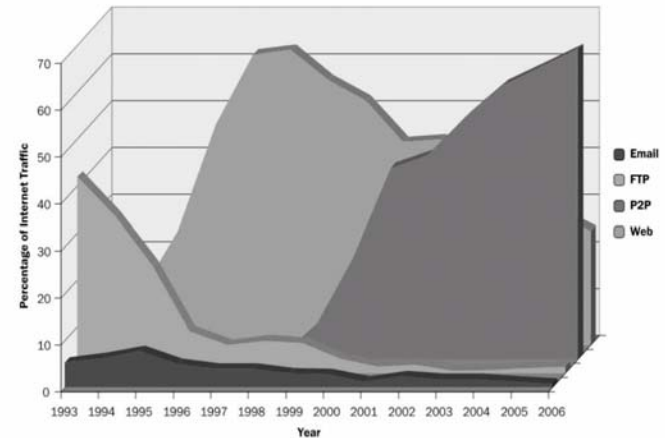
- This talk = „all“ I know about **today's peer-to-peer systems...**
- ... and a bit more! 😊
- Systems **evolve** over time, and hardly any client applies the same **algorithms**
- Thus:
  - focus on what I find interesting
  - some simplifications to focus on main **concepts**
  - selection of topics is **biased**
- Most importantly: when you know better, **let us know.**



# The Paradigm

- Key idea: Participating machines are both **consumers and contributors**

- Popularity: Peer-to-peer accounts for a large fraction of Internet **traffic**  
(source: CacheLogic.com)

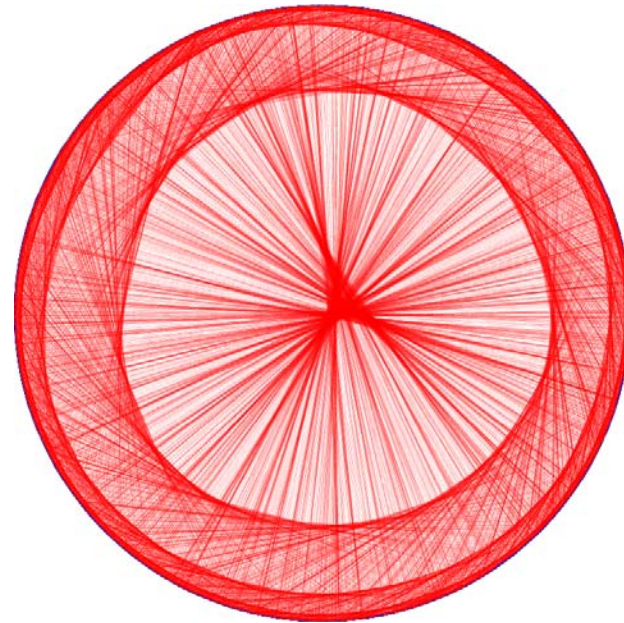


- Promises of the paradigm
  - Efficiency and **scalability**
  - **Robustness**, no single point of failure
  - Cheap: no expensive **infrastructure** at content distributor
  - „**Democratic**“ aspect: anyone can publish its contents / speeches / etc.

# From Theory to Practice... (1)

---

- Much **scientific literature** on peer-to-peer computing
  - Topics: scalability, dynamics / churn, heterogeneity, incentives, etc.
- Sample peer-to-peer systems (**mostly DHTs** in literature): who has heard of
  - Chord? Pastry? Tapestry? Kademlia?
  - Viceroy? Koorde?
  - SplitStream?
  - Pagoda?
  - etc.



# From Theory to Practice... (2)

- The **four evangelists**...
- If you read your average P2P paper, there are (almost) always four papers cited that “invented” efficient P2P in 2001:

Chord

CAN

Pastry

Tapestry

- These papers are somewhat similar, with the exception of CAN
- So what's the „**Dead Sea Scrolls of P2P**“?



## From Theory to Practice... (3)

---

„Accessing Nearby Copies of Replicated Objects in a Distributed Environment“, by Greg Plaxton, Rajmohan Rajaraman, and Andrea Richa, at SPAA 1997.

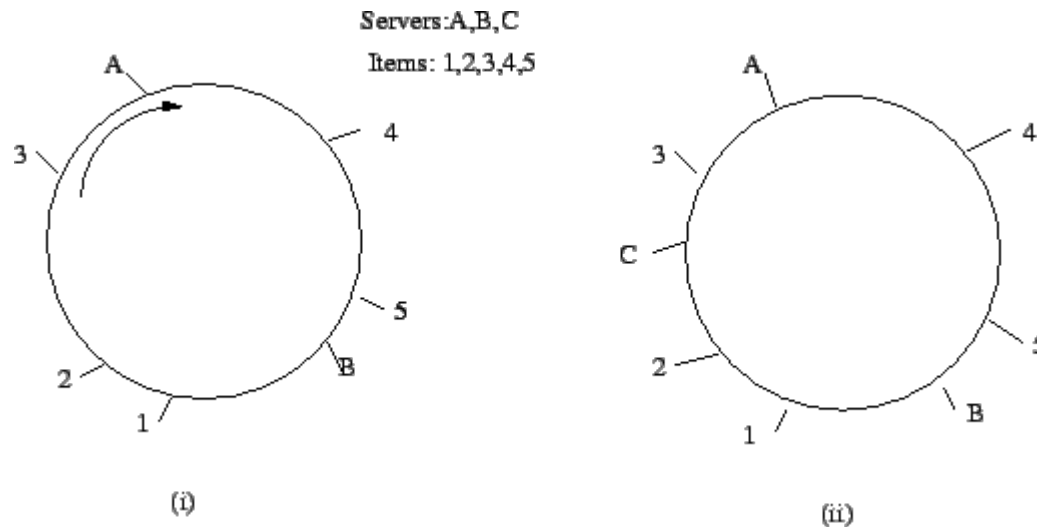
- Basically, the paper proposes an **efficient search** routine (similar to the evangelist papers). In particular search, insert, delete, storage costs are all **logarithmic**, the base of the logarithm is a parameter.
- However, it's a **theory paper**, so that alone would be too simple...
- So the paper takes into account latency; in particular it is assumed that nodes are living in a **metric**, and that the graph is of „bounded growth“ (meaning that peer densities do not change abruptly).

slide © Roger Wattenhofer



# From Theory to Practice... (4)

“Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web.” David Karger, Eric Lehman, Tom Leighton, Matthew Levine, Daniel Lewin and Rina Panigrahy, at STOC 1997.



- Big difference: still a **client/server** paradigm.

slide © Roger Wattenhofer

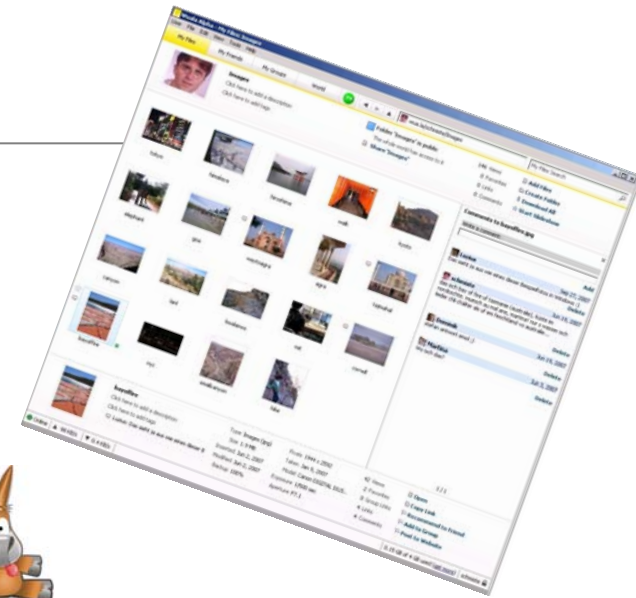




# From Theory to Practice... (5)



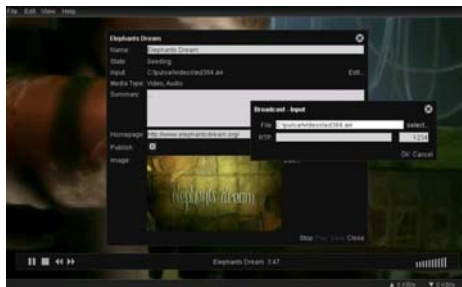
- Popular networks **in practice**: who has heard of
  - Skype?
  - Napster? Kazaa? eMule?
  - BitTorrent?
  - Kad network?
  - Zattoo?



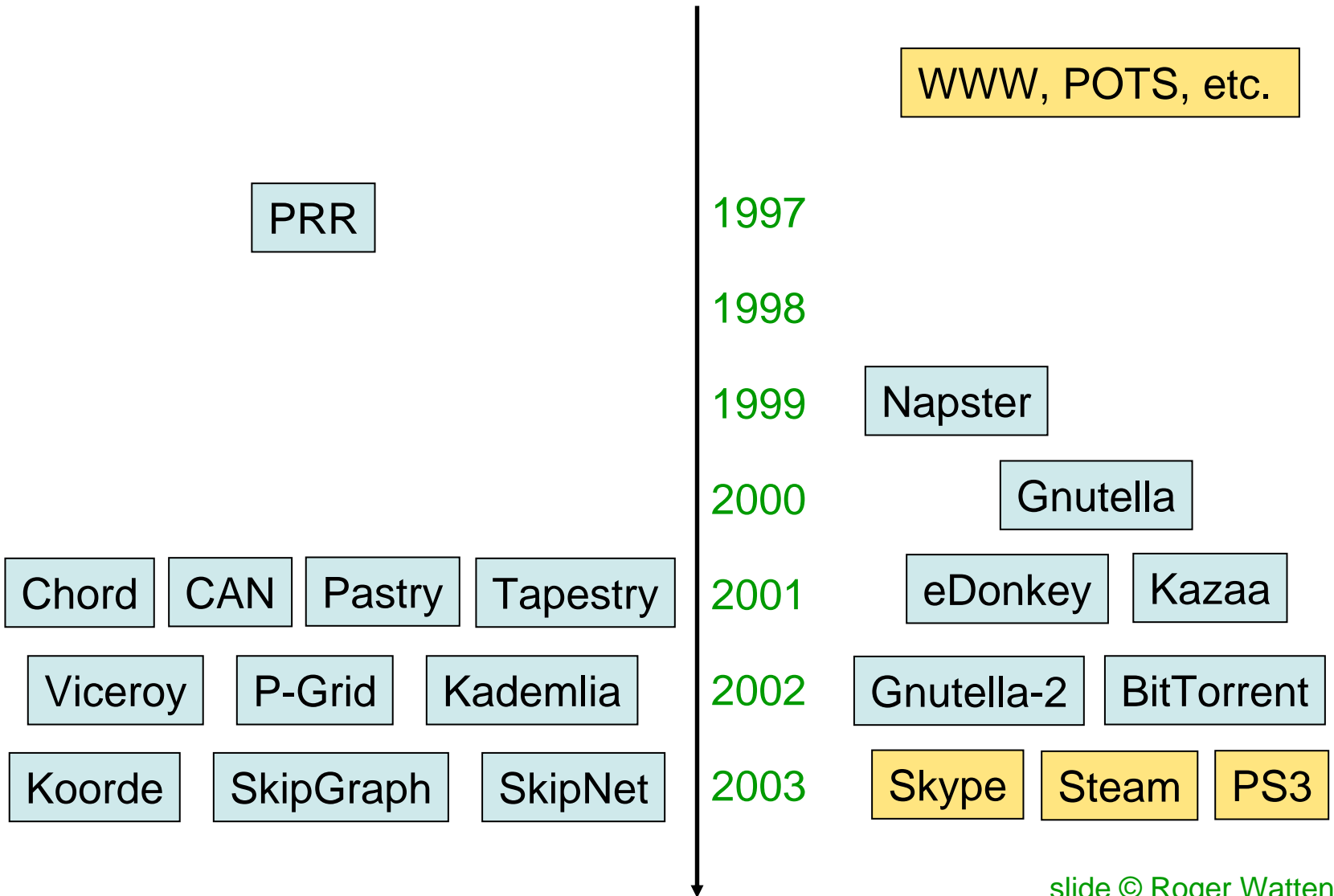
- Applications
  - Internet telephony
  - File sharing
  - TV streaming
  - Distribution of sw updates
  - etc.

IP	Client	T	Pieces	%	Down Speed
[161.111.167.163	Mainline 3.4.2	L		30.7%	41.4 kB/s
85.226.239.190	BitTornado 0.0.1	L		42.1%	44.7 kB/s
81.169.134.160	TorrentFlux	L		100.0%	4.8 kB/s
59.70.157.170	Mainline 3.4.2	L		29.5%	2.6 kB/s
195.132.185.221	Mainline 4.2.0	L		0.3%	1.3 kB/s
64.230.45.220	BitTornado 0.3.7	L		48.3%	3 B/s
66.60.29.24	Mainline 4.0.2	L		62.3%	1 B/s
85.74.173.83	Unknown 0[b----->]---,-E--]	L		5.3%	0 B/s
80.36.183.246	Mainline 4.2.0	L		39.1%	0 B/s
80.34.197.225	Azureus 2.3.0.6	L		100.0%	700 B/s
24.118.6.103	Azureus 2.3.0.6	L		100.0%	85 B/s
68.44.73.71	Azureus 2.3.0.6	L		100.0%	0 B/s
84.99.1.184	Azureus 2.3.0.6	L		30.7%	0 B/s
80.26.66.77	Azureus 2.3.0.4	L		15.9%	0 B/s
203.217.70.209	Azureus 2.3.0.6	L		61.6%	0 B/s
65.92.75.225	Azureus 2.3.0.4	L		100.0%	0 B/s
217.121.184.10	Azureus 2.3.0.6	L		100.0%	0 B/s
82.41.29.166	Azureus 2.3.0.4	L		100.0%	0 B/s
84.222.38.34	Azureus 2.3.0.4	L		100.0%	0 B/s
80.144.41.247	Azureus 2.3.0.6	L		100.0%	0 B/s
84.161.33.97	Azureus 2.3.0.6	L		100.0%	0 B/s
80.127.66.20	BitTornado 0.3.10	L		63.5%	8 B/s
63.230.53.111	BitTornado 0.3.7	L		28.5%	5 B/s
72.137.123.237	ABC 3.0.1	L		74.9%	1 B/s
212.195.125.66	ABC 2.6.9	L		61.5%	1 B/s
195.228.232.241	Mainline 3.4.2	L		61.3%	0 B/s

Azureus 2.3.0.6 803,141 Users {Nov 30, 17:52} IPs: 0 - 0/1/1 D: 100.3 kB/s U: [40K] 39.3 kB/s



# The Genealogy of Peer-to-Peer



slide © Roger Wattenhofer



# What happens behind the scenes of my peer-to-peer client?

The screenshot shows the Azureus peer-to-peer client interface. The window title is "Azureus" and the menu bar includes "File", "Transfers", "View", "Tools", "Plugins", and "Help". The toolbar contains various icons for file operations and network settings. The main window displays a list of peers under the "Peers" tab, with columns for IP, Client, T, Pieces, %, Down Speed, and Up Speed. The status bar at the bottom shows "Azureus 2.3.0.6", "803,141 Users", "{Nov 30, 17:52} IPs: 0 - 0/1/1", "D: 100.3 kB/s", and "U: [40K] 39.3 kB/s".

IP	Client	T	Pieces	%	Down Speed	Up Speed
161.111.167.163	Mainline 3.4.2	L		30.7%	41.4 kB/s	20.1 kB/s
85.226.239.190	BitTornado 0.0.1	L		42.1%	44.7 kB/s	8.3 kB/s
81.169.134.160	TorrentFlux	L		100.0%	4.8 kB/s	0 B/s
59.70.157.170	Mainline 3.4.2	L		29.5%	2.6 kB/s	10.1 kB/s
195.132.185.221	Mainline 4.2.0	L		0.3%	1.3 kB/s	0 B/s
64.230.45.220	BitTornado 0.3.7	L		48.3%	3 B/s	0 B/s
66.60.29.24	Mainline 4.0.2	L		62.3%	1 B/s	0 B/s
85.74.173.83	Unknown 0[-b----->]-,E--]	L		5.3%	0 B/s	0 B/s
80.36.183.246	Mainline 4.2.0	L		39.1%	0 B/s	0 B/s
80.34.197.225	Azureus 2.3.0.6	L		100.0%	700 B/s	0 B/s
24.118.6.103	Azureus 2.3.0.6	L		100.0%	851 B/s	0 B/s
68.44.73.71	Azureus 2.3.0.6	L		100.0%	0 B/s	0 B/s
84.99.1.184	Azureus 2.3.0.6	L		30.7%	19 B/s	0 B/s
80.26.66.77	Azureus 2.3.0.4	L		15.9%	0 B/s	0 B/s
203.217.70.209	Azureus 2.3.0.6	L		61.6%	0 B/s	0 B/s
65.92.75.225	Azureus 2.3.0.4	L		100.0%	0 B/s	0 B/s
217.121.184.10	Azureus 2.3.0.6	L		100.0%	0 B/s	0 B/s
82.41.29.166	Azureus 2.3.0.4	L		100.0%	0 B/s	0 B/s
84.222.38.34	Azureus 2.3.0.4	L		100.0%	0 B/s	0 B/s
80.144.41.247	Azureus 2.3.0.6	L		100.0%	0 B/s	0 B/s
84.161.33.97	Azureus 2.3.0.6	L		100.0%	0 B/s	0 B/s
80.127.66.20	BitTornado 0.3.10	L		63.5%	8 B/s	0 B/s
63.230.53.111	BitTornado 0.3.7	L		28.5%	5 B/s	0 B/s
72.137.123.237	ABC 3.0.1	L		74.9%	1 B/s	0 B/s
212.195.125.66	ABC 2.6.9	L		61.5%	1 B/s	0 B/s
195.228.232.241	Mainline 3.4.2	L		61.3%	0 B/s	0 B/s

It depends on the system.



VS



VS



VS

...

Some (simplified) examples!



# Napster:

One of the first and best-known  
„peer-to-peer“ systems



# Napster (1)

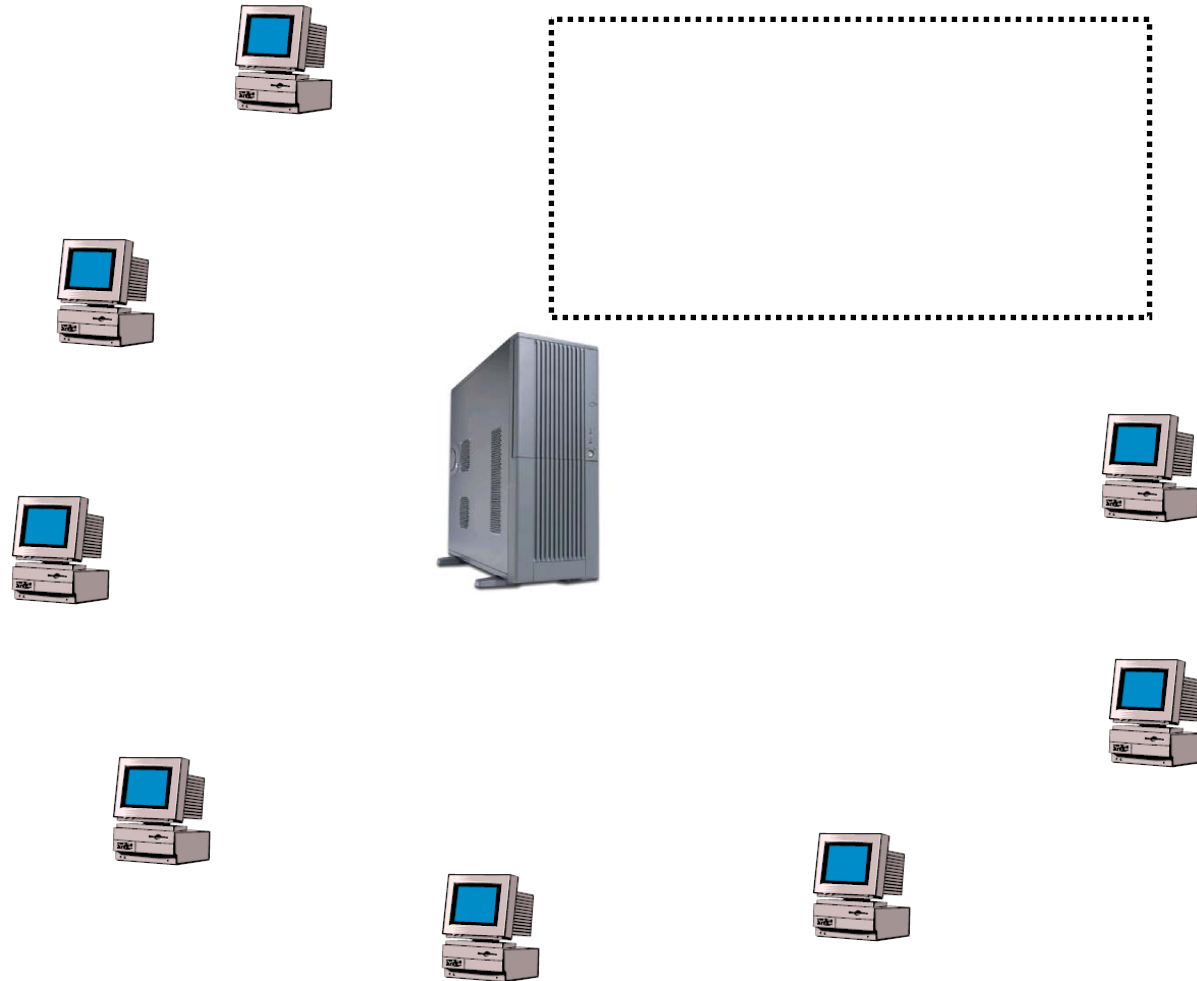
---

- One of the first „peer-to-peer“ **file sharing** systems (mainly MP3s)
  - Release year: 1999 (in the same year also first RIAA **law-suit**)
  - Shut down in year 2001 (today: pay service)
  
- Napster is **not a pure peer-to-peer system**
  - Relies on servers which store directory (but not files)
  - Resource discovery problem trivial: ask **index server**
  - Download then happens „peer-to-peer“ (not via server)

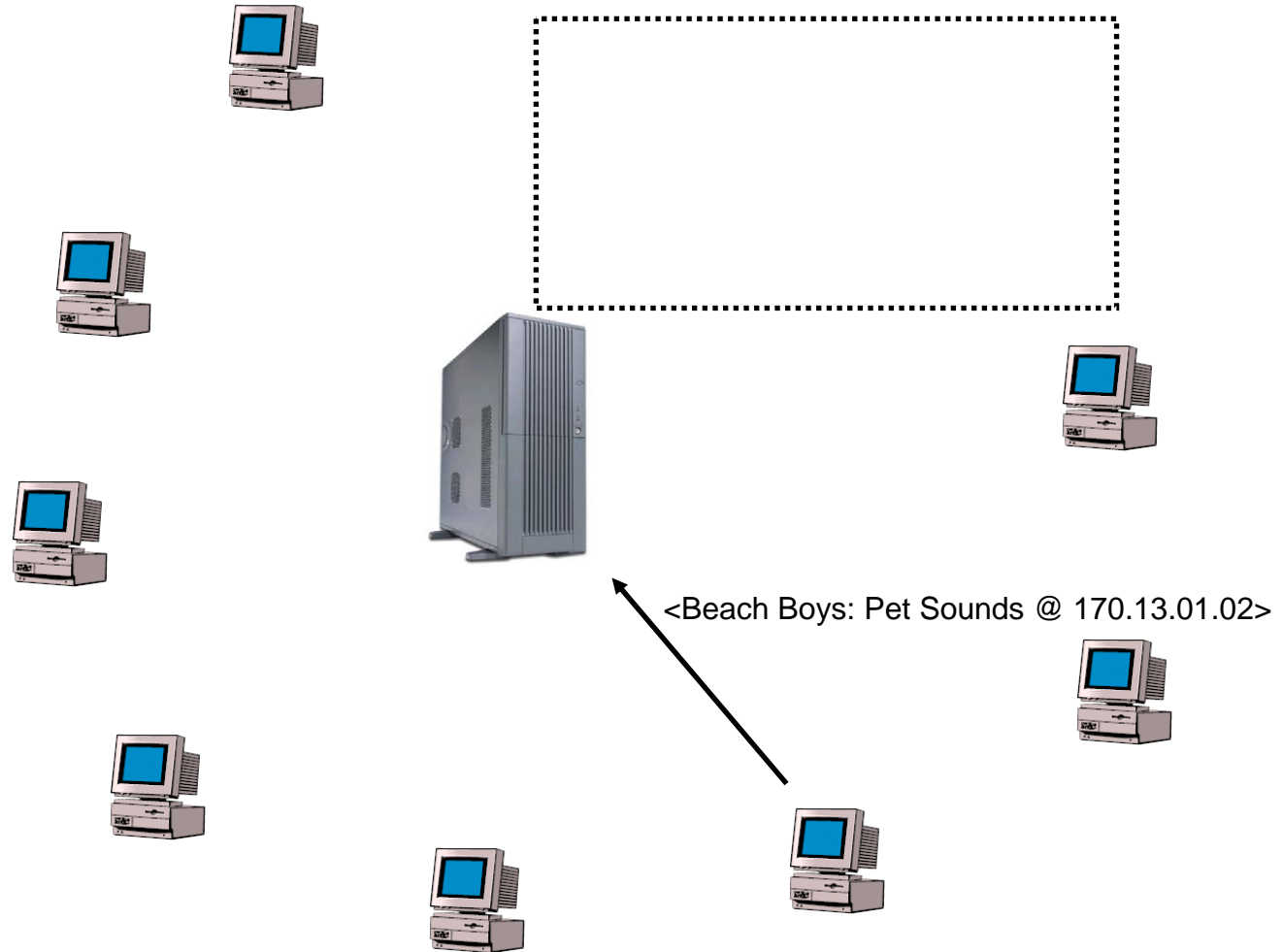


# Napster (2)

---



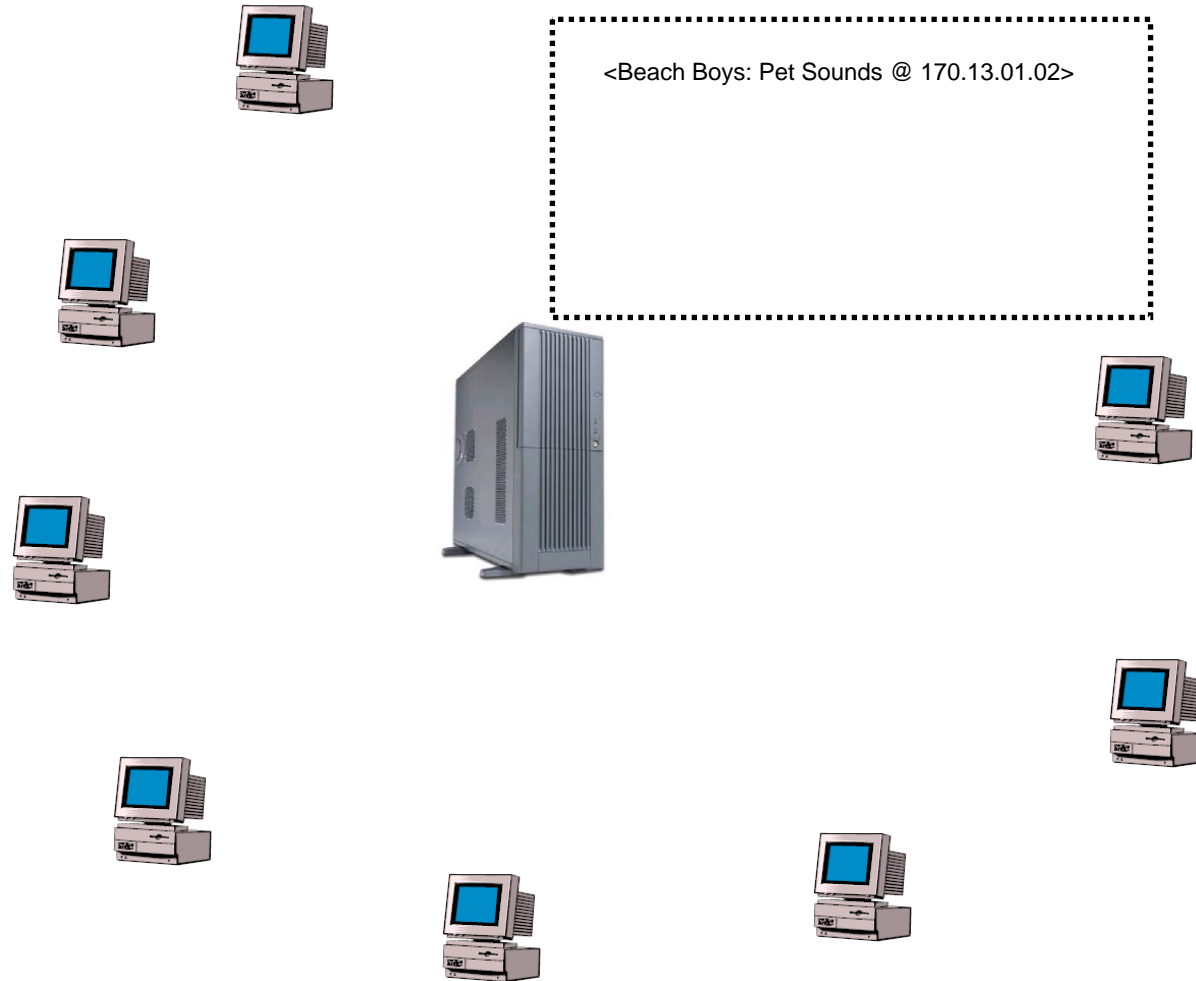
# Napster (2)



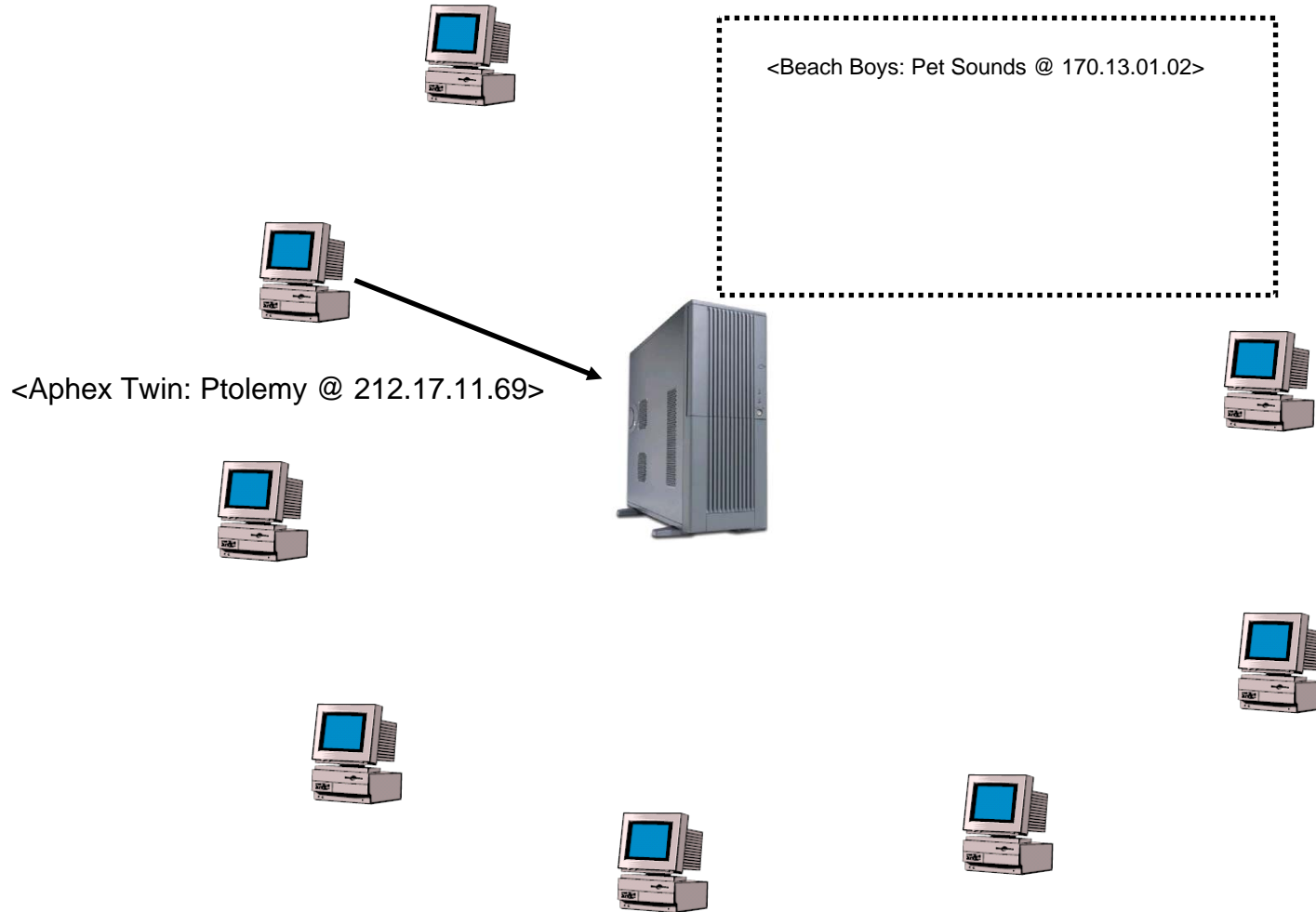


# Napster (2)

---

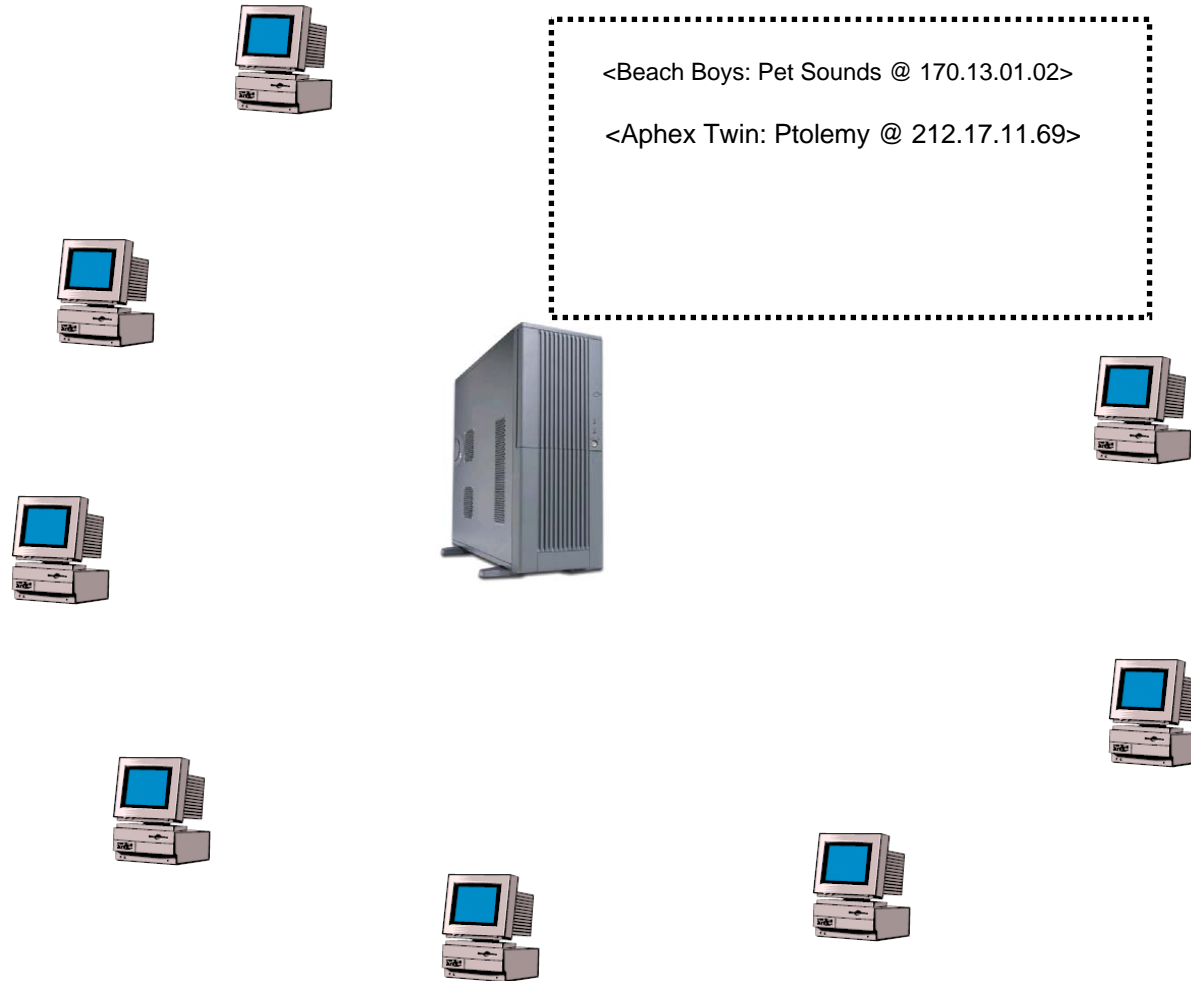


# Napster (2)

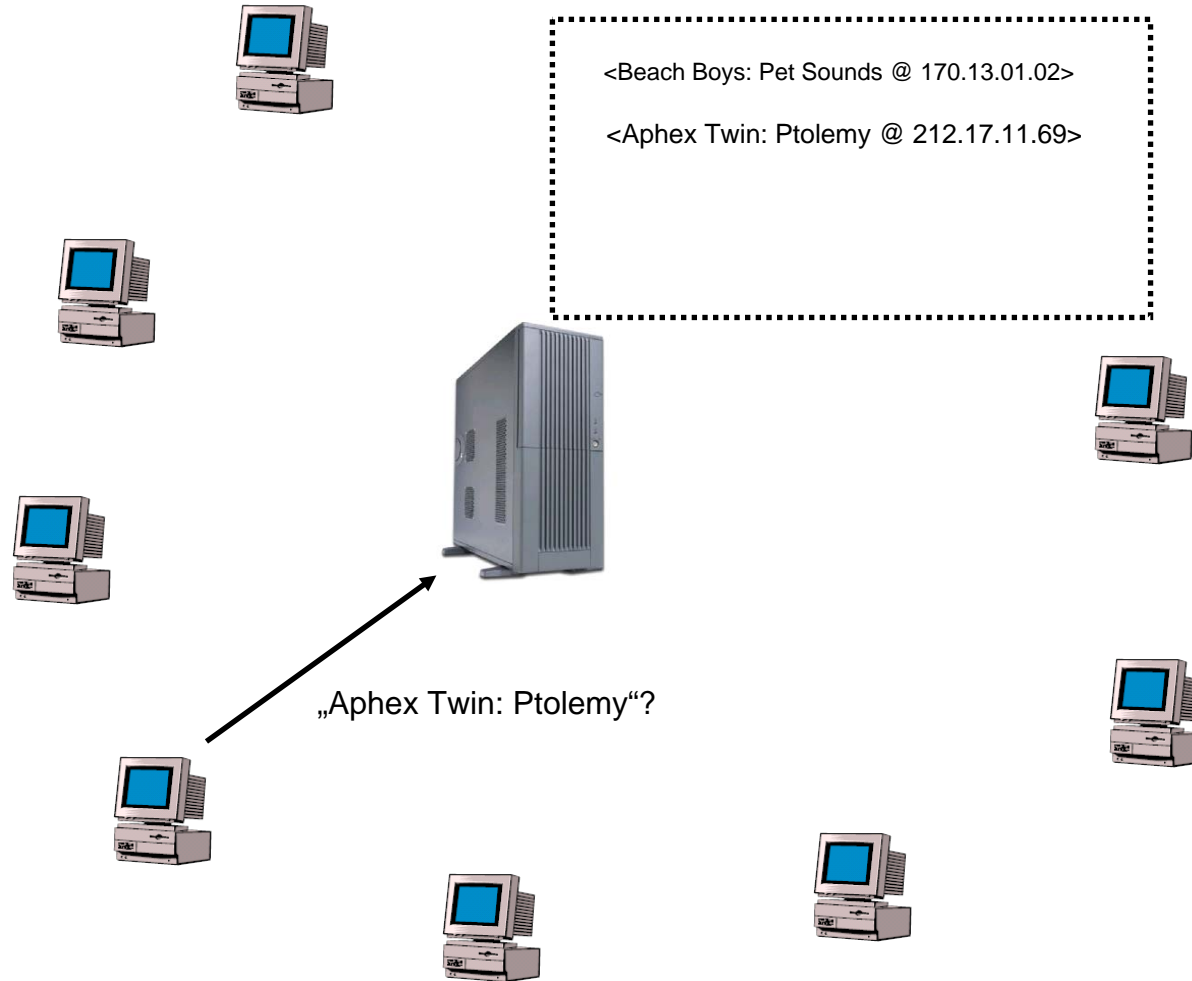


# Napster (2)

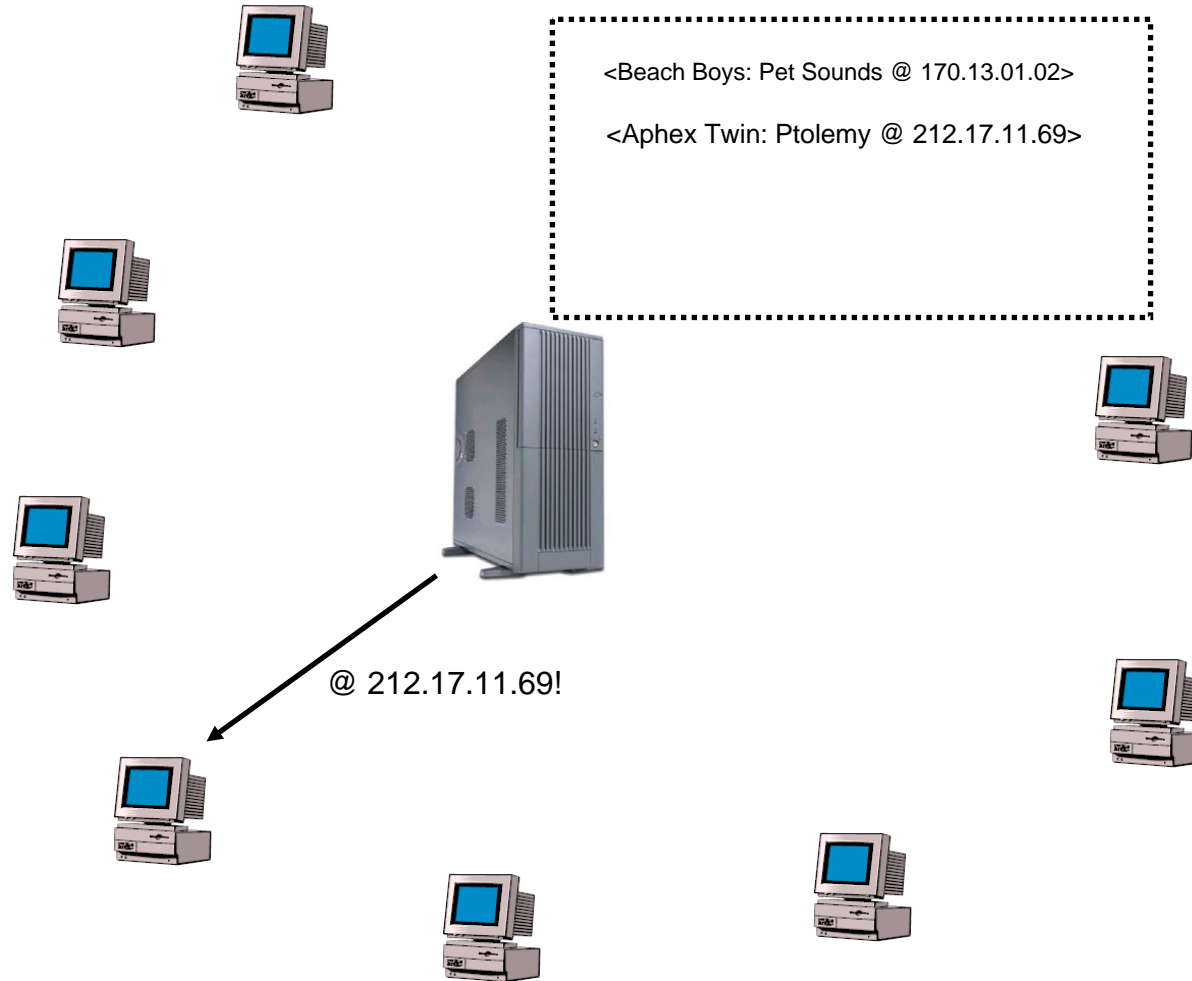
---



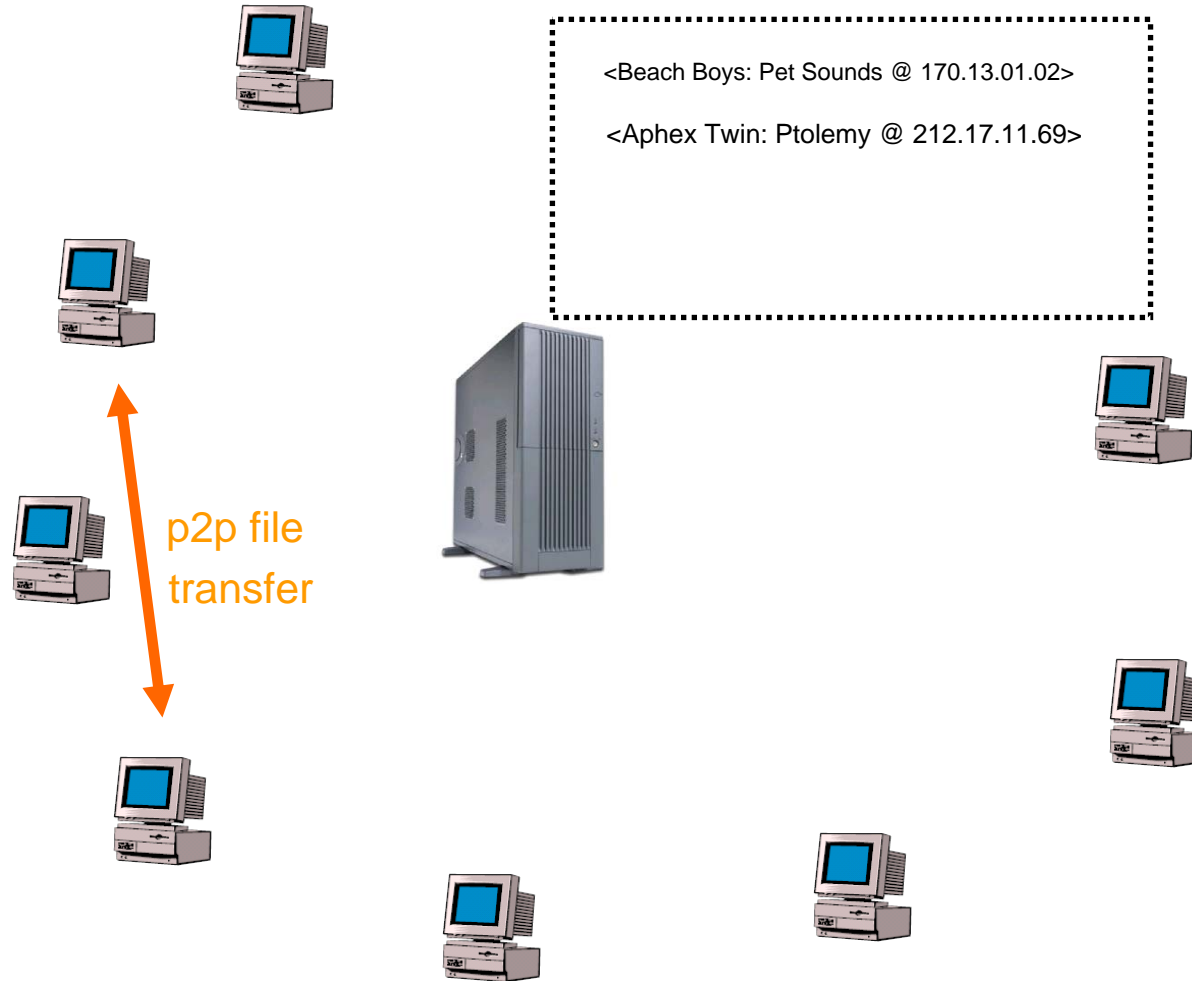
# Napster (2)



# Napster (2)



# Napster (2)



# Napster (3)

---

- Evaluation
  - Does the job: facilitates file sharing!
  - Highly popular
  - Not really peer-to-peer
  - Server = Single point of failure (legal action!)
  - Does not scale



# Gnutella:

An early, completely decentralized approach





# Gnutella (1)

---

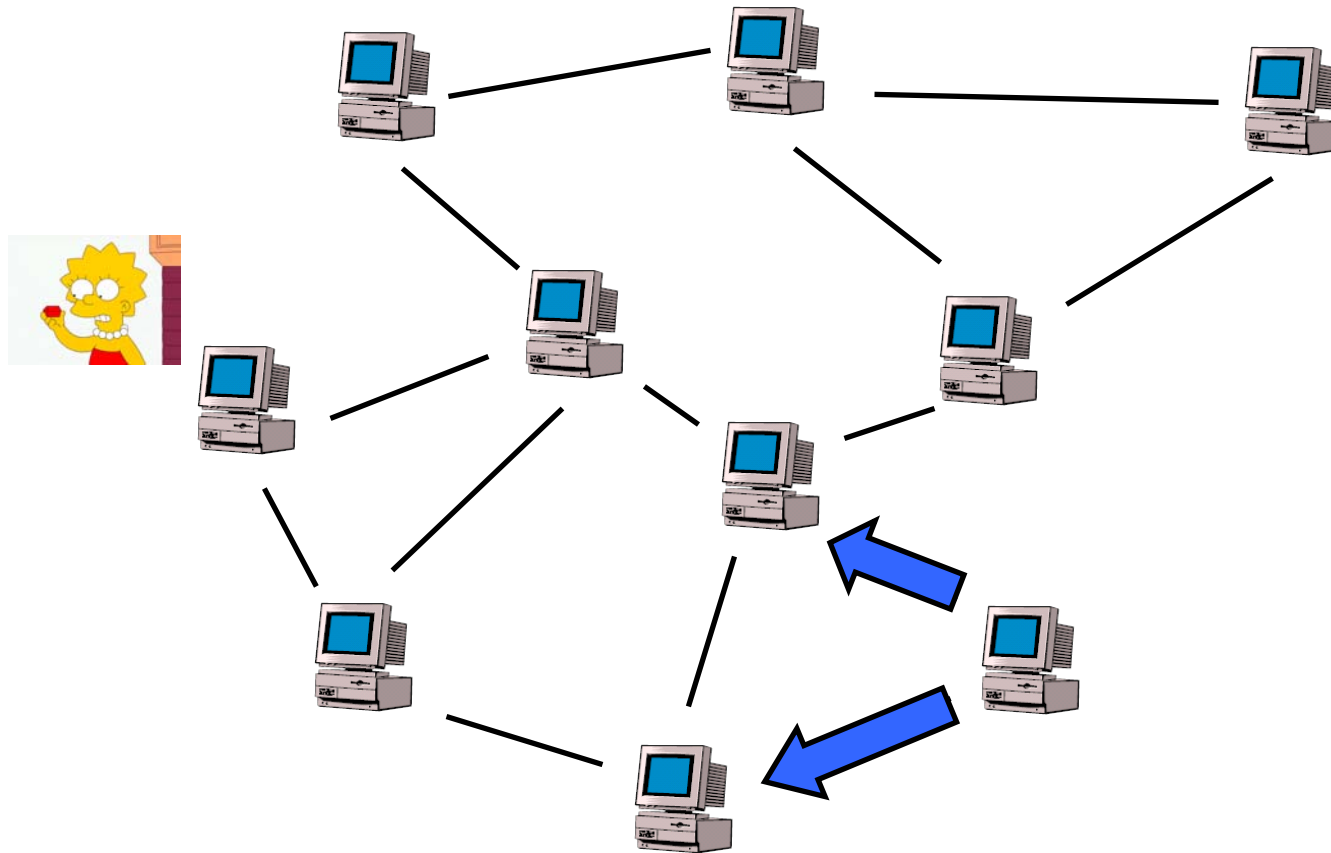
- Completely **decentralized** architecture
  - Beta release in March 2000
  - No index server!
  - Cannot be „shut down“
- Also very popular
  - Estimated 2+ million users
- Clients
  - **LimeWire**, BearShare, Acqlite, Mutella, ...
- Many Gnutella versions
  - Many different clients
  - **Protocol evolves** over time



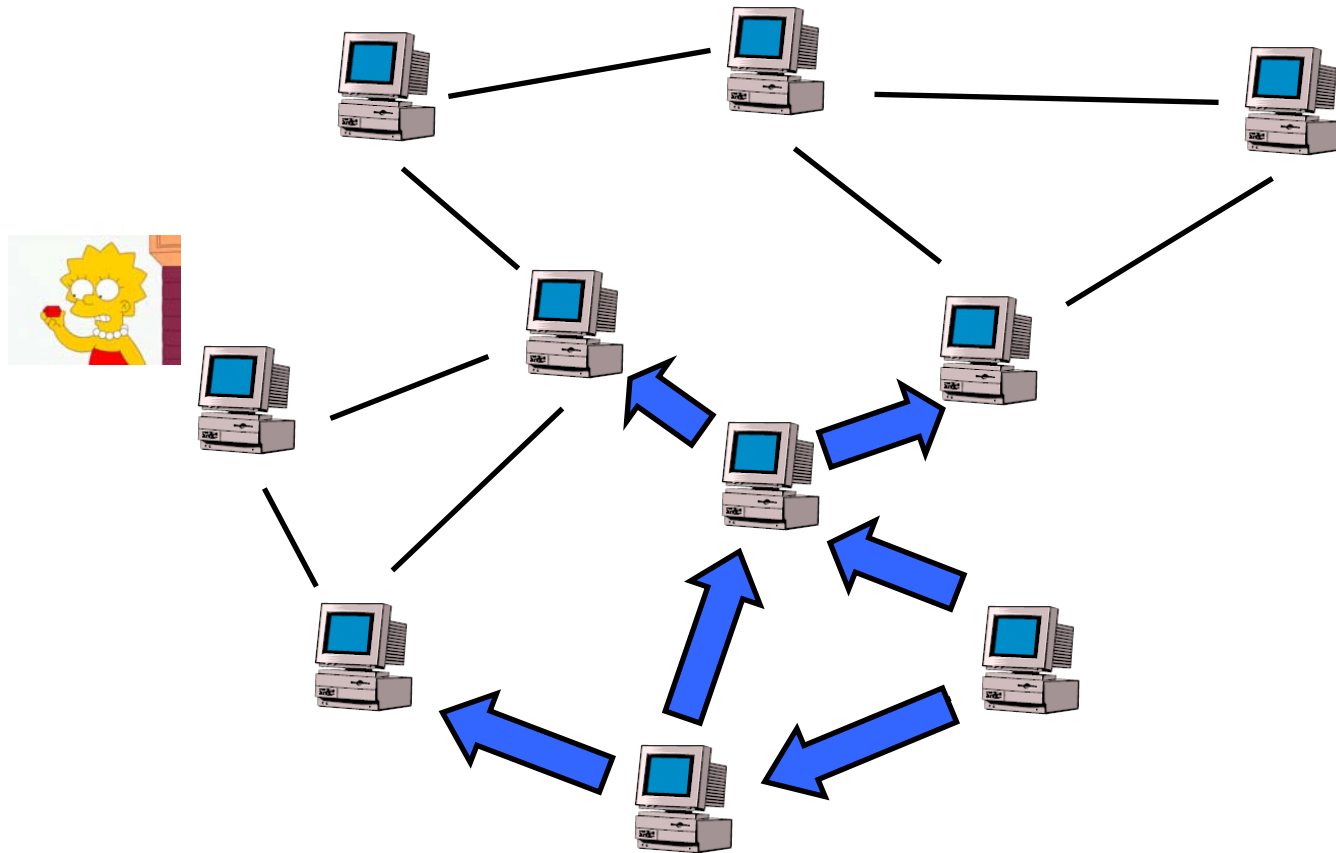
We will only present the main concepts of Gnutella!



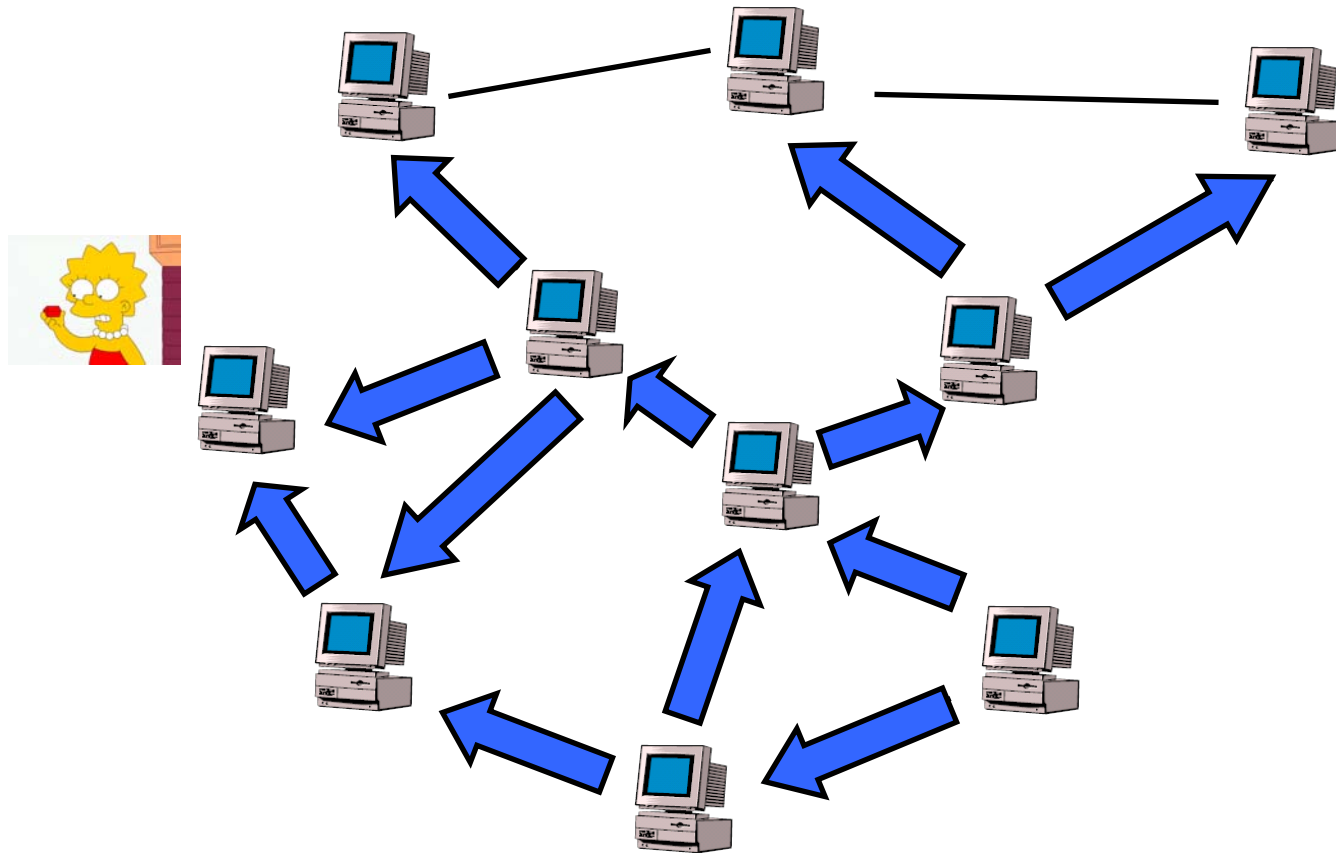
# Gnutella (2)



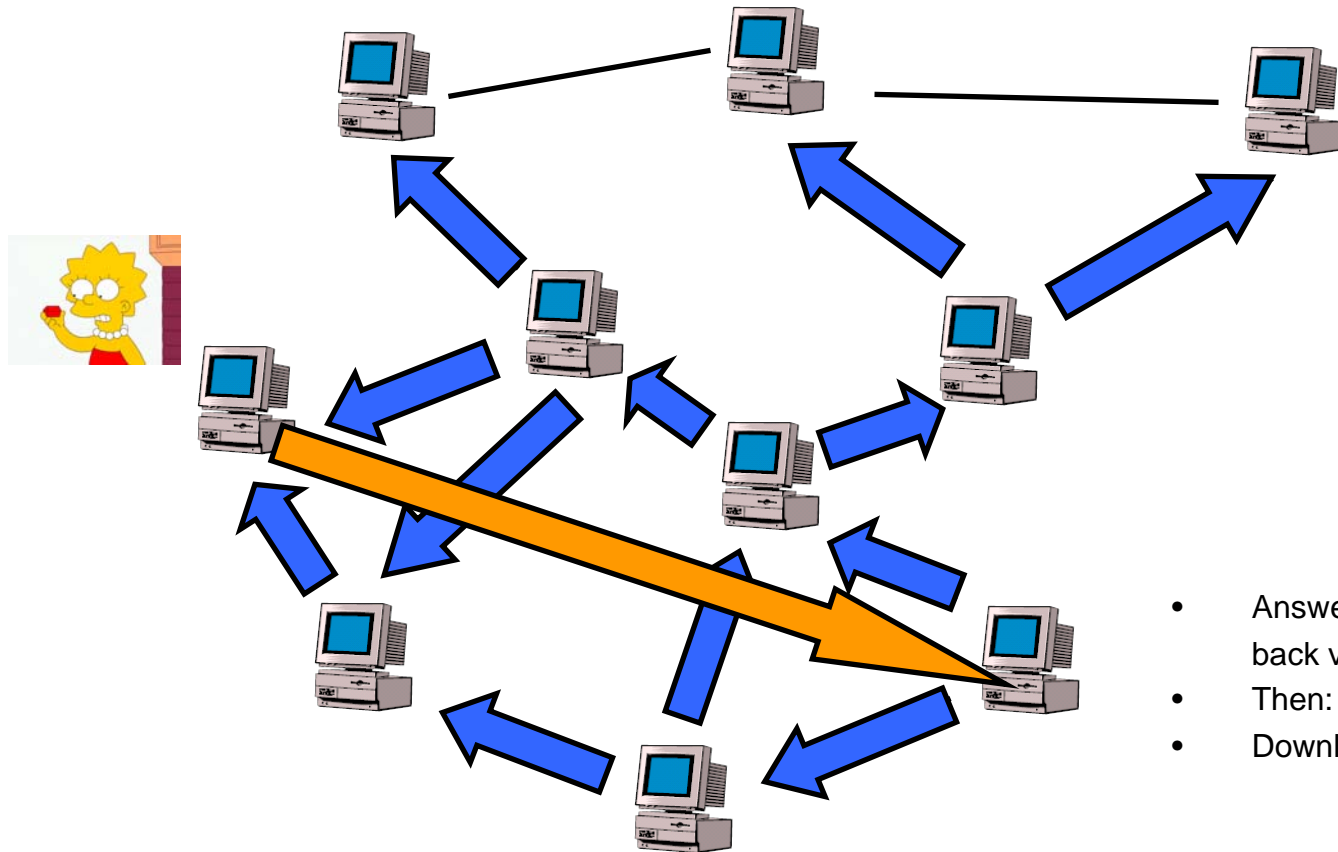
# Gnutella (2)



# Gnutella (2)



# Gnutella (2)

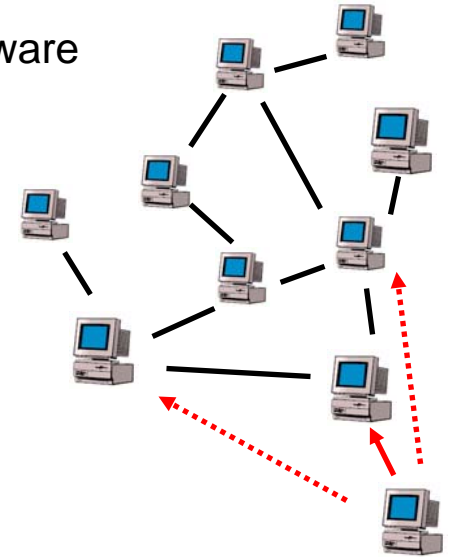


- Answers come back via multihop
- Then: direct download
- Download from one source



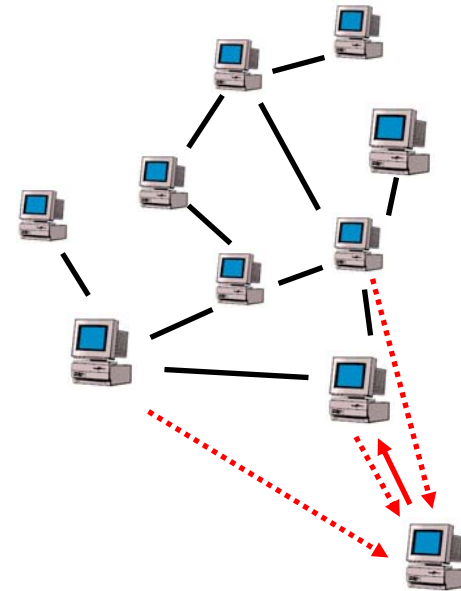
# Gnutella (3)

- Bootstrap
  - e.g., pre-existing address list of peers, **shipped** with the software
  - e.g., **web** caches
  - e.g., IRC **chat**
  - ...
- Topology
  - **join**: depends on client, no specific requirements
  - typically: starting with bootstrap peer, recursively explore neighbors until degree (depends on client) is reached
  - this can result in **inefficient** (redundant transmissions during flooding) or even **disconnected** topologies (unlike Napster)
  - countermeasure **high peer degree**?
  - some measurement studied found **small-world** / power law properties in modern graphs
  - after join, there is no rule how and when to find alternative peers for crashed neighbors
  - graph / out-degree distribution mainly a **social phenomenon**



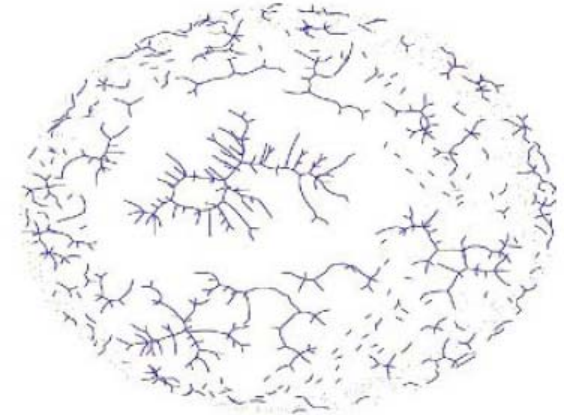
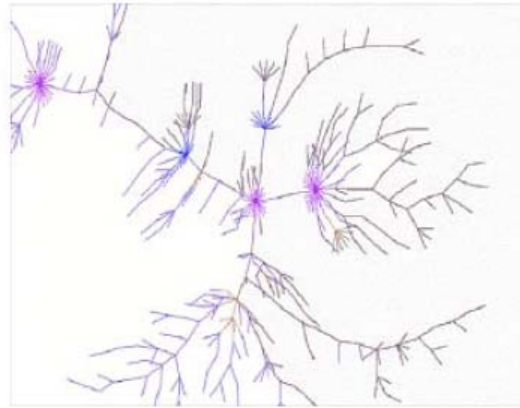
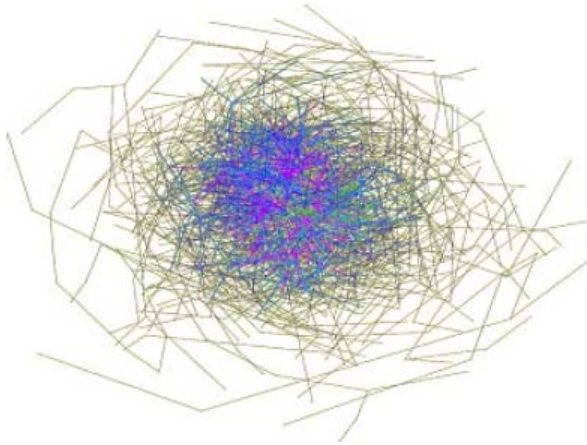
# Gnutella (4)

- The **ping/pong join** protocol
  - join operation similar to query operation
  - joining peer sends a ping message to neighbor
  - neighbor returns pong message, and **forwards** ping to its neighbors
  - **iteratively**: whenever a peer receives a ping, it sends pong to originator (multi-hop on **same path**)
  - up to some **time-to-live**
  - originator **randomly** selects subset of these peers as neighbors (neighborhood size:  $\geq 5$ )



# Gnutella (5)

---



- **Measurement study** 2001 with 1771 peers
  - „A Measurement Study of Peer-to-Peer File Sharing Systems“, 2002 (Saroiu, Gummadi, Gribble)
  - Left: Gnutella topology **Februar 16, 2001**
  - Middle: 30% peers removed at random (still large connected component)
  - Right: 4% highest degree peers removed
  - quite robust to **random faults**, but not **worst-case faults** (attacks)



# Gnutella (6)

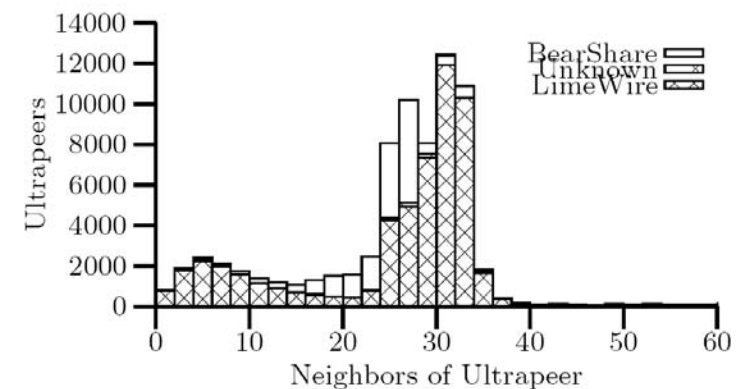
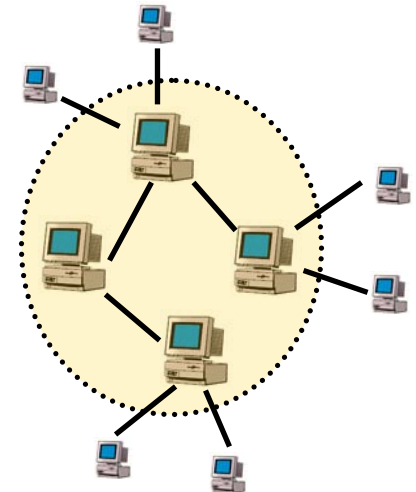
---

- Evaluation
  - Fully decentralized and „simple“
  - Hardly any **restrictions** on topology...
  - ... but hardly any **guarantees** (e.g., diameter or connectivity) either
  - Still not very scalable: flooding yields many **redundant transmissions**
  - In fact, when Napster was unplugged, Gnutella **broke down** due to the inrush of former Napster users
- Files **may not be found** although they exist (unless entire network is flooded)
  - Problematic for „rare files“
  - But approach directly supports queries like **range queries**, Boolean queries, etc.



# Gnutella (7)

- Many extensions (e.g., **Gnutella-2**), e.g., hybrid **two-tier architecture**
  - **Ultrapeers**: have higher bandwidth, do most of the routing
  - Ultrapeers form the „**core network**“, are connected to (many) other ultrapeers; store **indices** of their leaves
  - Ideally, an ultrapeer has a high bandwidth, large session times, and other peers can connect to it via TCP
  - Ultrapeer **degree**: around 30 (LimeWire)
  - **Leaves**: only connect to a small number of ultrapeers
  - Renders system more efficient in heterogeneous environment
- Search by **dynamic flooding** on core network
  - increasing TTL, until around 100 results are found
- Peers **decide themselves** which role they assume (no control)



# BitTorrent: Cooperation in swarms



# BitTorrent

---

- Peer-to-peer computing **relies on the contributions** of the peers
  - However, peers may be selfish!
  - How to provide **incentives for cooperation**?
- Simple solution: **tit-for-tat**
  - **Barter system**: peer  $p$  offers resources to peer  $p'$  while  $p'$  offers resources to  $p$
  - But: What if  $p'$  is not interested in the resources (e.g. files) of  $p$ ? (cf **real economy**)
  - BitTorrent heralded **paradigm shift**: it showed that cooperation can be achieved **on a single file**
- Main ideas
  - Peers interested in a certain file form a **swarm**
  - Swarms can be found via **trackers**
  - Instead of sharing the entire file, file is divided into smaller **pieces**
  - Pieces of a file are exchanged in a **tit-for-tat** like manner (details later)
  - **Pipelining**: peer downloads different parts of the file from different sources **concurrently**



# BitTorrent Architecture



website with .torrent file



visit website

Bootstrap: .torrent meta files are offered by various websites and can be found, e.g., via web search engines



# BitTorrent Architecture

---



website with .torrent file

download .torrent file



# BitTorrent Architecture

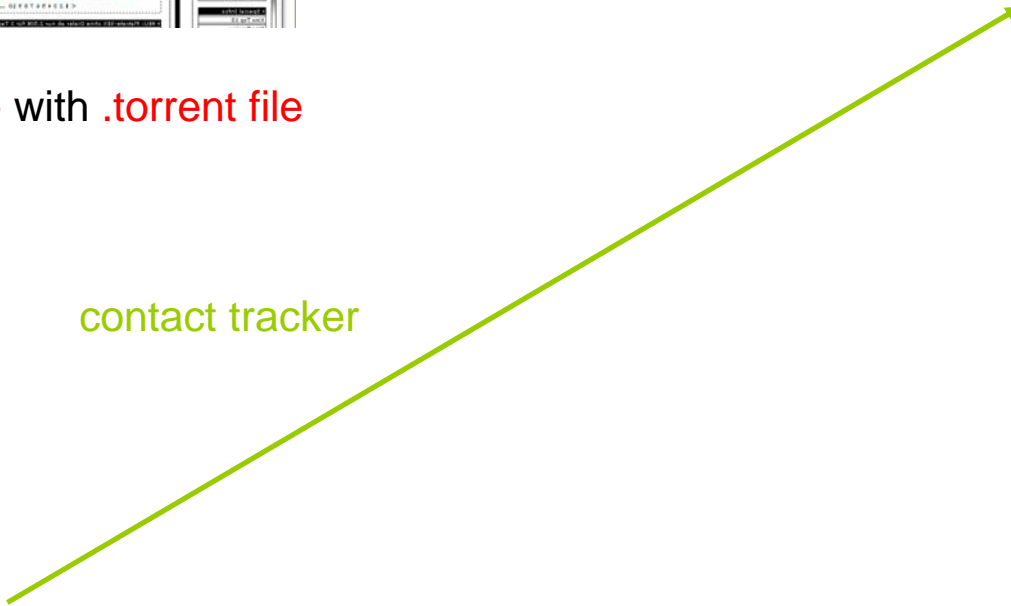


website with .torrent file

Tracker



contact tracker

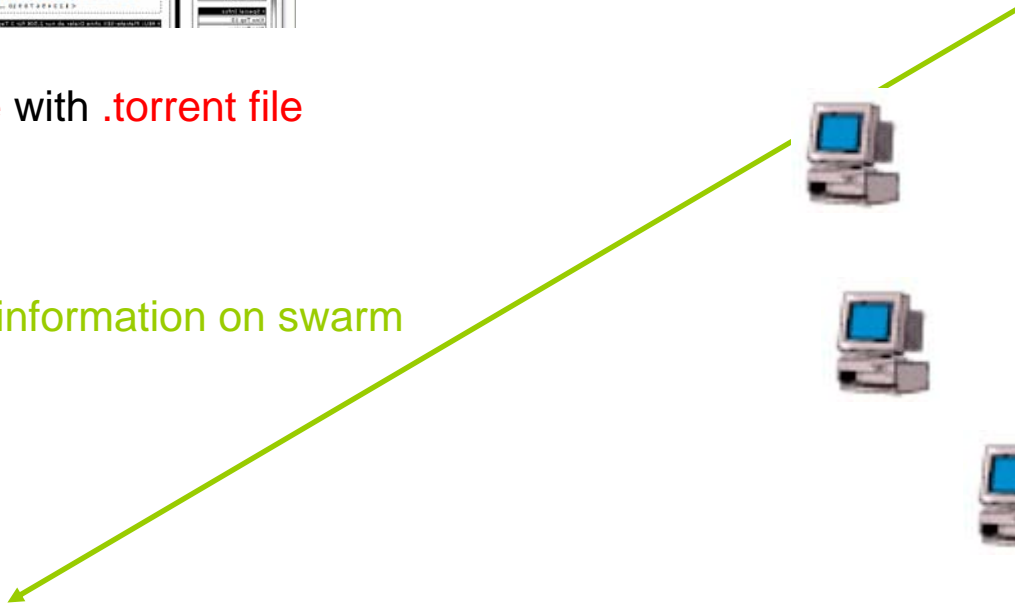


# BitTorrent Architecture



website with .torrent file

retrieve information on swarm



Tracker





# BitTorrent Architecture



website with .torrent file

Tracker



join swarm



Cache / FIFO: connect to most recent set,  
not structured / hypercubic etc.



# BitTorrent Architecture

---

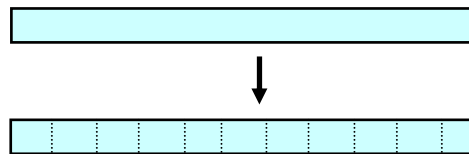
- **Tracker** maintains information on peers in swarm
  - „problematic“: tracker knows **many IP addresses**
  - easy to check whether these peers are really downloading...
- Peers send **periodical updates** to the tracker about their status
  - e.g., all 30 minutes
  - peers also contact the server when they **join and leave**
- When joining, peers establish **roughly 40 connections** to other peers in swarm
- If number of responsive neighbors falls below 20 connections, tracker is contacted again
  - peer retrieves **additional contacts**



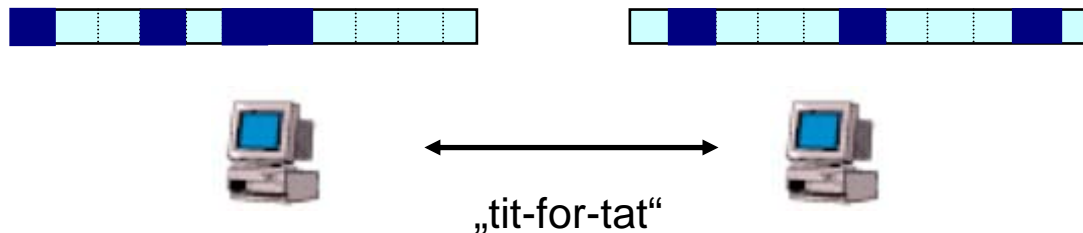
# BitTorrent Swarm

---

- Swarm consists of peers interested in **same file** (or collection of files)
- File is divided into several **pieces** (usually a couple of thousand pieces)



- Peers **trade** these pieces („swarming“)  
- In a **tit-for-tat** like manner



# BitTorrent: Peer Types

---

- Peers in the swarm which have all pieces are called **seeders**



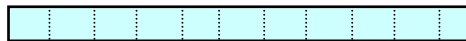
- Peers which only have a subset of all pieces are called **leechers**



# BitTorrent: Bootstrap Problem

---

- But what about **newly joined peers**?
  - Do not have anything (any pieces) to offer...
  - Will not be able to trade!
  - That's known as the **bootstrap problem**



- That's (probably) the reason that BitTorrent does not employ a pure tit-for-tat policy: concept of **optimistic unchoking**

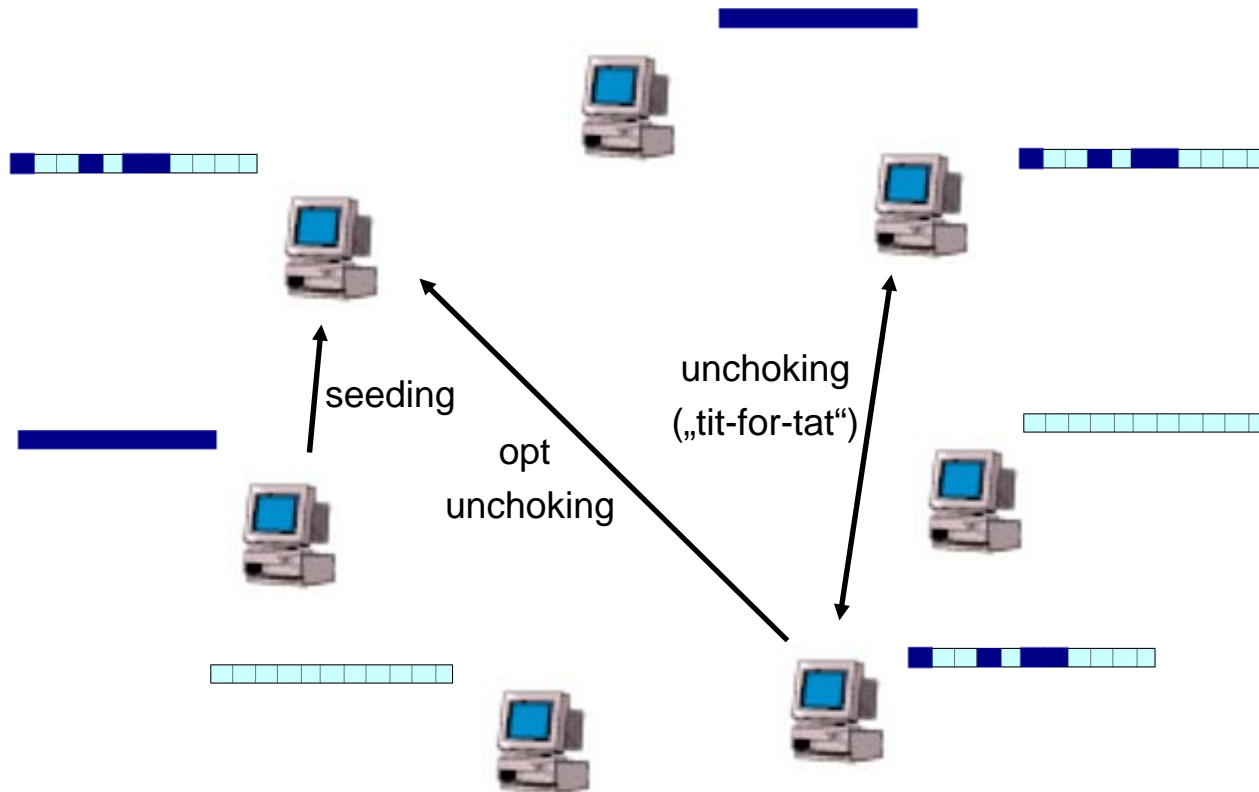


# BitTorrent: Incentive Mechanism

---

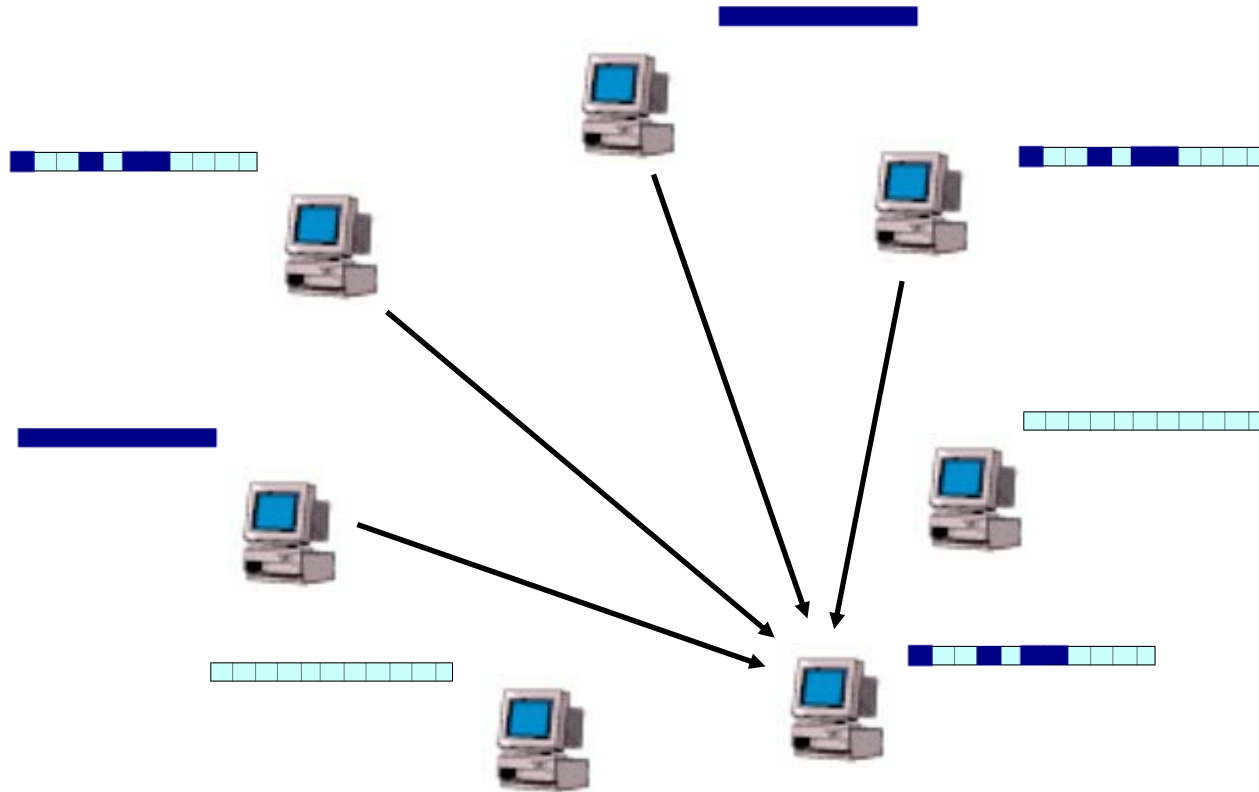
- BitTorrent uses the following mechanism
- **Seeders** upload their pieces to leechers in a **round robin** fashion
  - round robin = „one after another“
- **Leechers** perform a modified version of tit-for-tat, and solve the bootstrap problem by using **optimistic unchoking** slots
- Concretely, leechers do the following:
  - Peers upload concurrently to the „best neighbors“ (**active set**)
  - Active set typically consists of **5 peers** only
  - We say that active set is „**unchoked**“
  - Peer uploads (as much as possible) to peers in active set (not purely tit-for-tat)
  - Download rate received from neighbors is **evaluated every 10 secs**
  - In addition, a peer **optimistically unchokes** a random neighbor: in uploads pieces for free to this neighbor for roughly 30 secs, independently of the download received; gives that peer **a chance** to **bootstrap** or to become an **active** set peer!

# Swarm Overview



# Concurrent Downloads

---





# Local Rarest First Policy

---

- A peer is informed about the new pieces available at its neighbors
  - „**have-message**“
- Which piece should a peer download?
- Typical policy: **LRF**
  - Local rarest first
  - Try to download piece which is least replicated **among neighbors**
  - Minimizes chance that rare piece gets lost when seeder leaves
- Exception: Pieces are selected at random until **first piece** is completely downloaded, enables a fast start (rare pieces can typically only be obtained from one, potentially slow, peer)
- Thus, since pieces are retrieved in random order (**non-contiguous download**), BitTorrent is not directly made for, e.g., **on-demand streaming** where pieces at the beginning of the file should be downloaded earlier



# BitTorrent Download Characteristics

---

- BitTorrent downloads differ from, e.g., **HTTP downloads**
  - HTTP more or less constant speed from the beginning
  - BitTorrent uses many TCP sockets
- Download performance slow in the **beginning** (takes time to **collect neighbors** and **sufficient data** to become effective uploader)
- Full speed during „**midgame**“
- **Endgame** slower again: only a small number of pieces left to download, restricted choice of neighbors offering this content
  - BitTorrent uses **special endgame mode** where the same subpieces are requested **in parallel** and redundantly from several neighbors in order to remain efficient towards the end (if a subpiece is obtained from one peer, **cancel** is sent to others)



# Data Verification and Subpieces

---

- In practice, pieces (size ~100 KB) are further divided into **subpieces**
  - Pipelining: More pending requests, improves TCP throughput
  - Schedule new request whenever subpiece arrives
  - Parallelism (subpieces from different peers)
  - Subpieces of a piece can be obtained from different peers (some clients restrict to one peer after some time)
- The **.torrent metafile** contains checksum for each piece (but not subpiece)
  - SHA1 hashing algorithm
  - Most BitTorrent clients **ban IP address** if verification fails



# Evaluation of Fairness Mechanism (1)

---

- Cooperation is **important** in p2p computing
  - incentives needed if people are selfish
  - **measurement studies**: large fraction of peers are free-riders
- BitTorrent is one of the first systems to tackle this problem algorithmically
- Other approaches
  - e.g., **Kazaa client**: monitors its contributions
  - can be bypassed by implementing different client (**Kazaa Lite**) which hardwires contribution level to maximum
  - many other solutions (e.g., **virtual money** systems)
  - some proposals are real economies almost, have to deal with **inflation**, deflation etc. => complex!
  - BitTorrent one of the practically most relevant strategies...



# Evaluation of Fairness Mechanism (2)

---

- BitTorrent works very well in practice and is a **huge success**
- **Cheating** is still possible though
  - e.g., clients such as **BitThief** or BitTyrant
  - poses interesting algorithmic problems (cf also **game theory**)



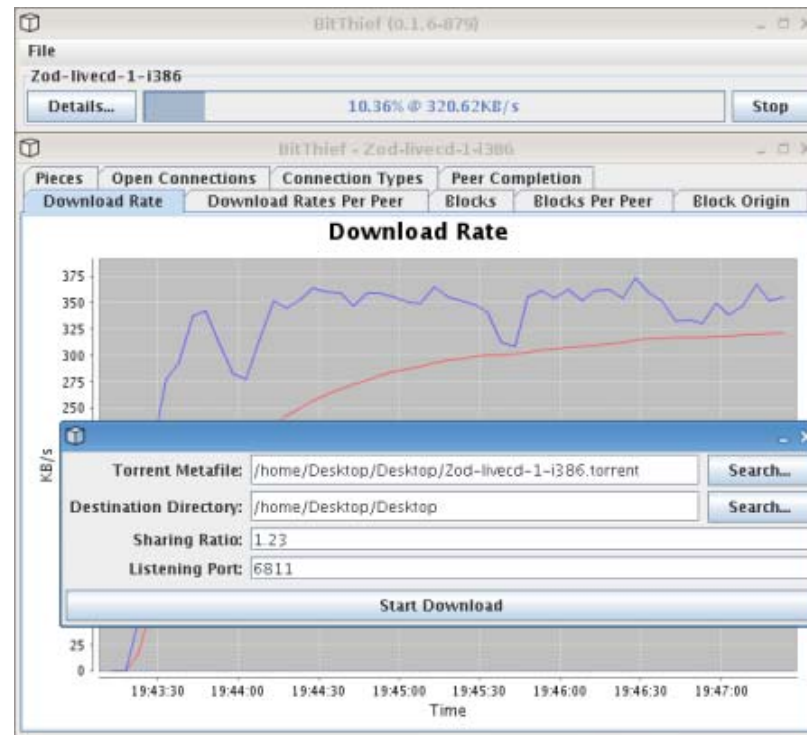
# Evaluation of Fairness Mechanism (3)

---

- How to cheat?
- Peers can **re-contact tracker** more frequently (=> more neighbors)
- More neighbors => benefit more frequently from optimistic unchoking slots (**free-ride!**)
- **Sharing communities**: BitTorrent networks which require user registration, monitor contribution of users; peers can **announce wrong upload rates** to tracker and benefit from more seeders
- Active set: peers can behave strategically and upload just enough to become member of **active set**, not more
- etc.

# Example: BitThief Client (1)

- **BitThief** is a **Java** client (implemented **from scratch**) which achieves fast downloads without uploading **at all**



## Example: BitThief Client (2)

---

BitThief's three tricks:

- Open as **many TCP connections** as possible (no performance problem!)
- Contacting tracker again and again, **asking for more peers** (never banned during our tests!)
- **Pretend** being a great uploader in **sharing communities** (tracker believed all our tracker announcements)

=> Exploit optimistic unchoking

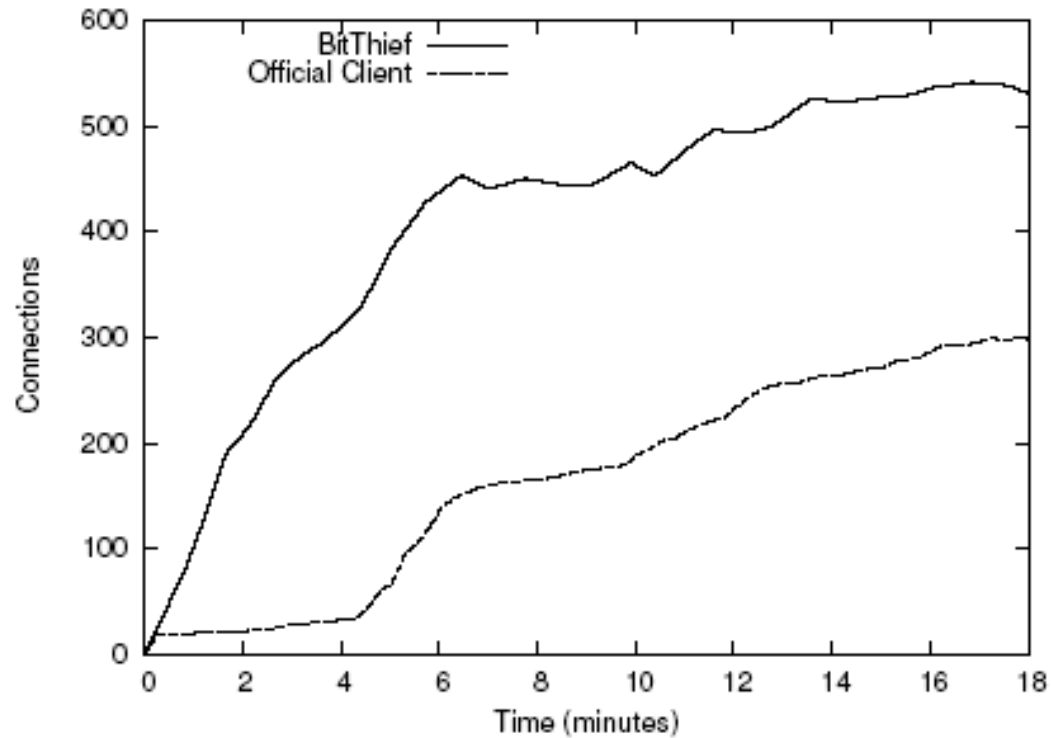
=> Exploit seeders

=> Exploit sharing communities



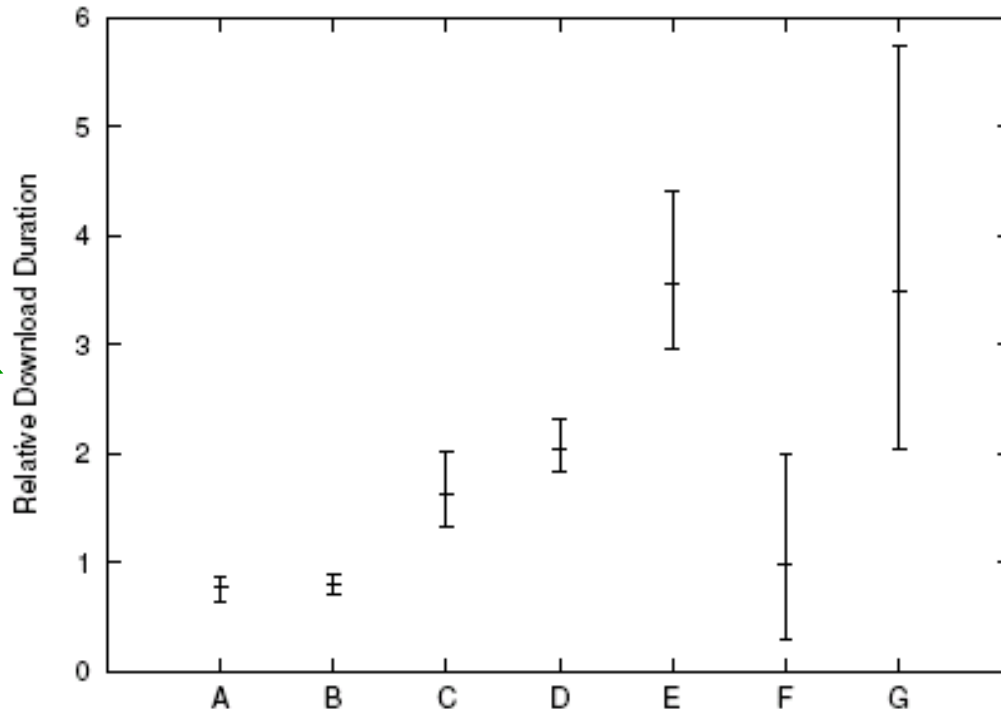


# Example: BitThief Client (3)



# Example: BitThief Client (4)

2  
compared to  
official client  
(with unlimited  
number of  
allowed  
connections)



4  
BitThief with public  
IP and open  
TCP port

number of peers  
announced  
by tracker

max  
peers found  
by BitThief

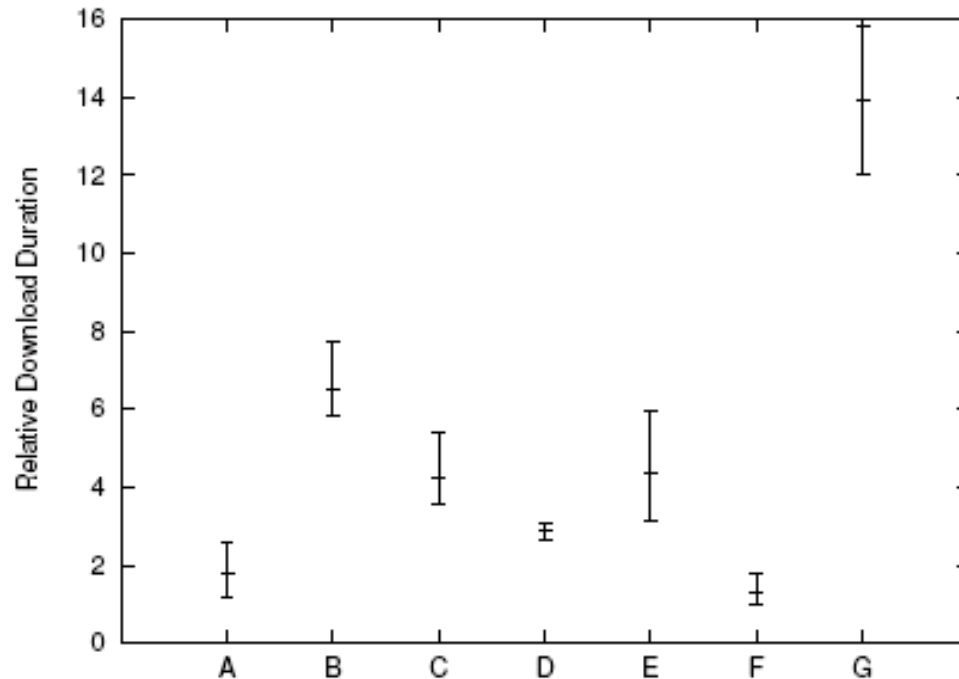
- 3
- With seeders...
  - **All downloads finished**
  - Fast for **small files** (fast startup),  
**many peers** and **many seeders!**

1

	Size	Seeders	Leechers
A	170MB	10518 (303)	7301 (98)
B	175MB	923 (96)	257 (65)
C	175MB	709 (234)	283 (42)
D	349MB	465 (156)	189 (137)
E	551MB	880 (121)	884 (353)
F	31MB	N/A (29)	N/A (152)
G	798MB	195 (145)	432 (311)



# Example: BitThief Client (5)



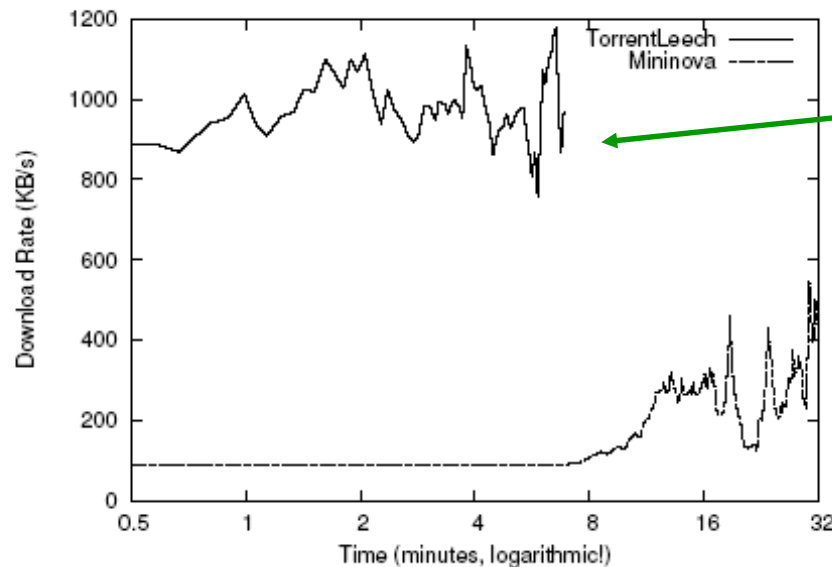
- Without seeders...
- Seeders detected with bitmask / have-message
- Even **without seeder** it's fast
- Unfair test: **Mainline client** was allowed to use seeders

	Size	Seeders	Leechers
A	170MB	10518 (303)	7301 (98)
B	175MB	923 (96)	257 (65)
C	175MB	709 (234)	283 (42)
D	349MB	465 (156)	189 (137)
E	551MB	880 (121)	884 (353)
F	31MB	N/A (29)	N/A (152)
G	798MB	195 (145)	432 (311)



# Example: BitThief Client (6)

- **Sharing communities** ban peers with low sharing ratios
- Uploading is encouraged; user registration required
- Client can report uploaded data itself (**tracker announcements**)
  - as tracker does not verify, it's **easy to cheat**



4 x faster!  
(BitThief had a faked sharing ratio of 1.4; in both networks, BitThief connected to roughly 300 peers)



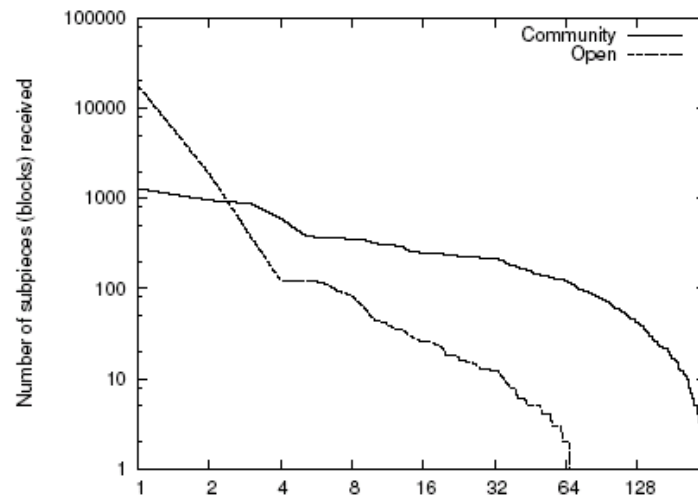
# Example: BitThief Client (7)

All information available to the tracker comes from the periodic announce messages peers send to it:

Tracker HTTP Request

```
GET /announce?...&uploaded=86016&downloaded=22528&left=81920&...
```

- In communities, contribution is **more balanced**
- Reason?
  - Peers want to **boost ratio**? Users more **tech-savvy**? (**less firewalled** peers? **faster** network connections?)



## Example: BitThief Client (8)

---

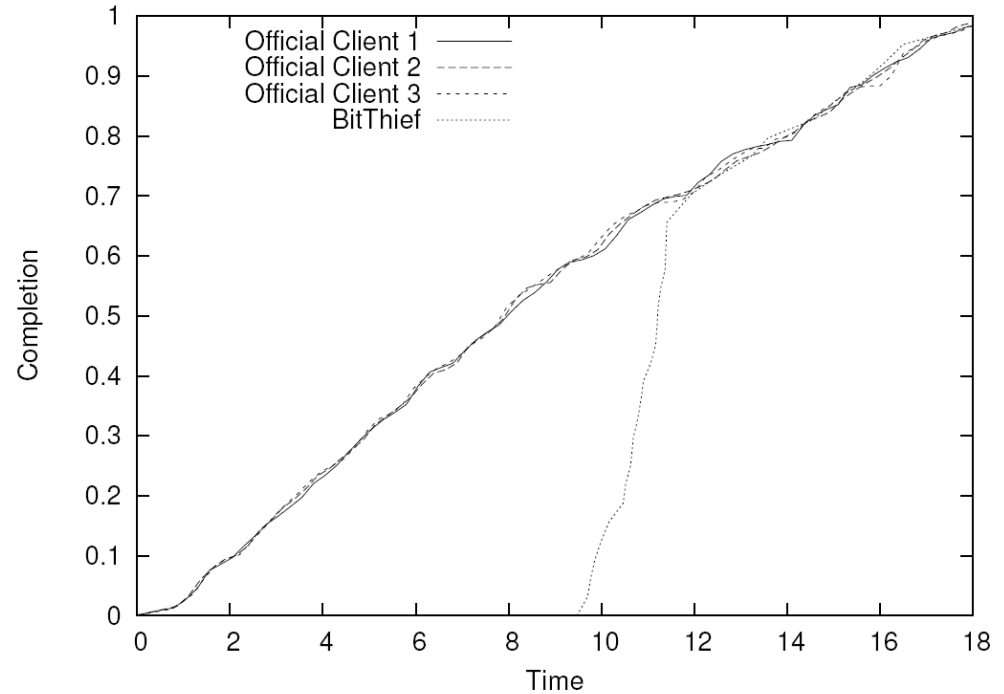
Some tricks did not work for BitThief:

- Announce many **available pieces** (0%-99% all the same, 100% very bad, considered a seeder)
- **Upload garbage** (easier with mainline client than with Azureus; Azureus remembers from which it has got most subpieces/blocks and tries to get all from him; otherwise you are banned)
- **Sybil attacks** with same IP address (goal: more often in „round robin unchoke slots“ of seeder)
- ...



# Example: BitThief Client (9)

- Particularly fast if
  - **Many seeders**
  - Sharing **communities**  
(many and fast seeders!)
  - **Small files**: Aggressive startup behavior of BitThief
  - Few and **slow seeders**: Other leechers are starving, plenty of redundant „optimistic unchoking slots“, BitThief relatively good

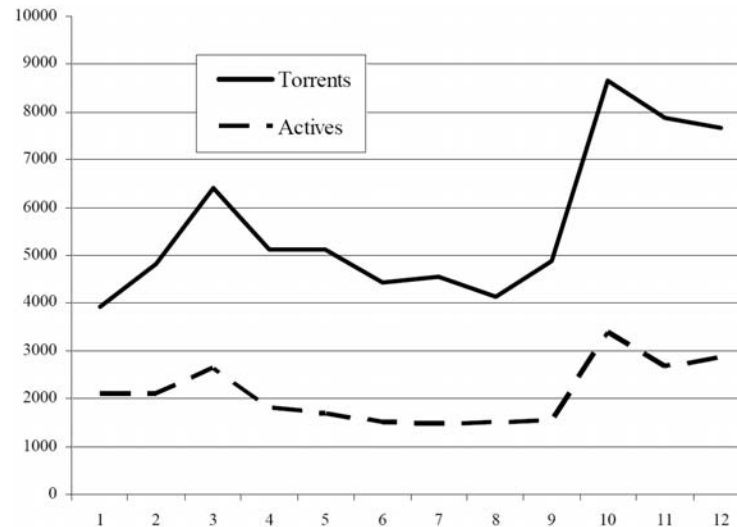


- Relatively slow if
  - Few fast seeders
  - Seeders are occupied, other leechers also busy with tit-for-tat



# Example: BitThief Client (10)

- Are people **selfish**?
  - no advertisement of client
  - poor GUI (will change now...)
  - collects data...





# Example: BitTyrant

---

- **BitTyrant** (Piatek et al., NSDI'07)
  - Another **strategic** BitTorrent client
  - Goal: more efficient downloads, **uploading allowed**
- Means: smart **neighbor selection**
  - **e.g.**, client seeks to be among top 5 neighbors (active set) **at minimal cost**
  - BitTyrant has larger active set
  - find peers with **good reciprocation ratio** ...
  - ... i.e., peers which upload much but need little
  - etc.



# Future Fairness Mechanisms?

---

- How to improve BitTorrents robustness to **selfish attacks**?
- Problem: Strict tit-for-tat impossible due to **bootstrap problem**
- Recent proposal: **fast extension**
  - newly joined peers also obtain „**venture capital**“ for free
  - i.e., pieces which can be downloaded from other peers without reciprocating
  - however, peer p only obtains **random subset** of pieces
  - this subset depends on p's IP address (from all peers the same)
  - in absence of seeders, free-riding is no longer possible!



# Final Remarks

---

- BitTorrent is still a **centralized** peer-to-peer system
  - introduces vulnerability
  - e.g., websites hosting trackers can be shut down (e.g., suprnova.org etc.)
- In 2005, a **distributed tracker protocol** has been released
  - e.g., for torrents which do not have a working BitTorrent tracker
  - Azureus is **Kademlia DHT** (see later), not compatible with official DHT
  - unfortunately, not much information available...
  - e.g., find new peers even without tracker
  - e.g., efficiently find rare missing pieces during end game?



# The eMule Client and Kad: Towards distributed hash tables



# Towards Distributed Hash Tables

---

- Seen so far:
  - **Napster** = server-based p2p architecture
  - **Gnutella** = unstructured p2p architecture
  - **BitTorrent** = swarms of peers interested in same file, tracker-based
- Recently, distributed hash tables and **structured p2p systems** also emerge in practice
- A case study of **eMule**...



# eMule & eDonkey

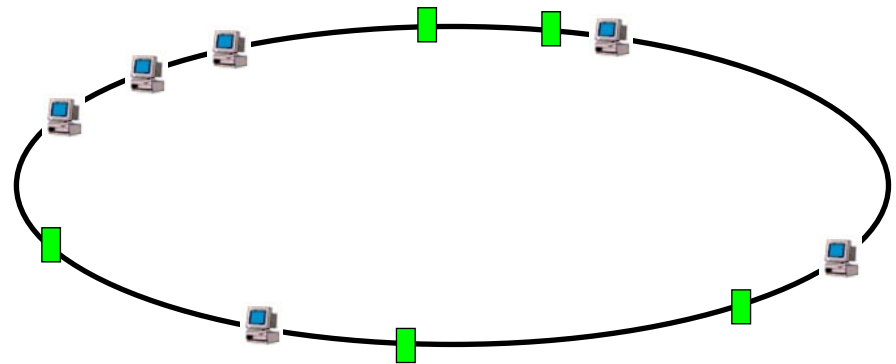
---

- The **eMule client** allows to connect to two different networks
  - the server-based **eDonkey2000 (eD2K)** network
  - the decentralized **Kad network**
  - open-source, and many mods exist...
- **eDonkey2000 network**
  - **popularity**: several million users
  - not very interesting from algorithmic point of view (server-based)...
  - eMule is connected to a eD2K server
  - at login time, client informs about available files
  - client maintains a file with a list of servers (in order of acquaintance)
  - most servers are based on **lugdunum software** (not open-source)
  - client iterates from one server in the list to the next until **roughly 300 results** have been collected
  - concentration on „popular“ servers, problematic when taken down (e.g. Razorback 2.0)?
- **Kad network based on DHT**
  - in more detail now...

# DHT Refresher (1)

---

- „Distributed hash table“
  - Peers and data have **overlay IDs** (or keys)
  - E.g., peer ID is hash of peer's IP address
  - E.g., file ID is hash of file name or file content



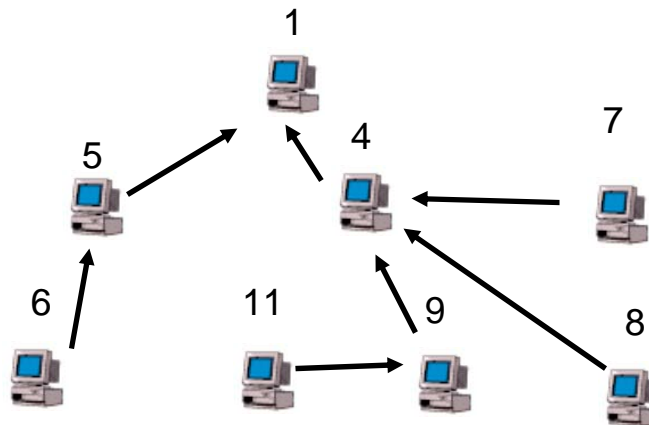
- Typically, both IDs are chosen from same space
  - e.g., 1-dimensional  $[0,1)$  space, data is stored on „closest peer“ (**consistent hashing** approach)
  - Peers are connected to each other with respect to their IDs (**structured peer-to-peer topology**)



## DHT Refresher (2)

---

- Data can be found efficiently (also **rare data**)
  - Routing algorithms beyond „**flooding**“ and **random walks**
- Overlay topology gives **guarantees**
  - simple rules ensure **connectivity** and **low diameter**
  - networks often **hypercubic**
  - e.g.: peers have unique numbers as identifiers
  - rule „connect to a peer of lower ID“ already ensures connectivity





# DHT Refresher (3)

---

- Some common mechanisms and principles...
- Mechanism 1: Do not store entire files at the corresponding position, but **only the pointer**
  - Can be copied much more quickly
  - Beneficial under dynamics
  - Nobody has to store other people's files
- Directed search and routing:
  - for a DHT lookup, you need to know the **file hash** / file key



# DHT Refresher (4)

---

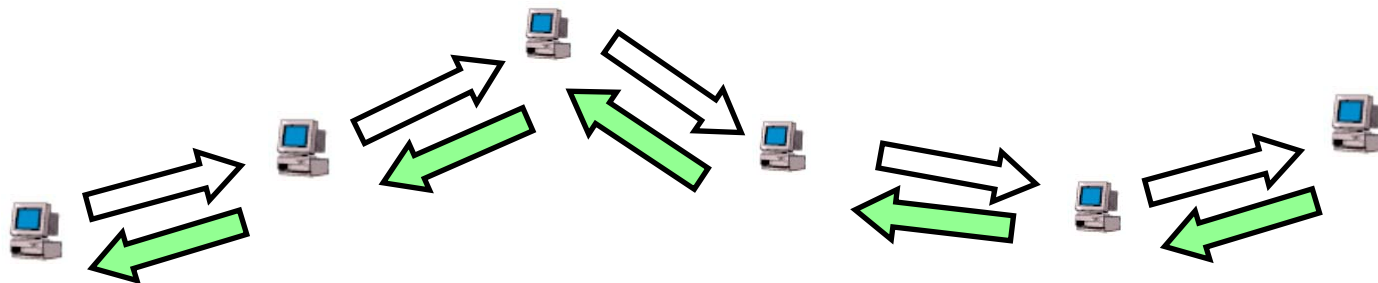
- In order to find a file, a peer needs the **file hash**
- Mechanism 2: introduce **another indirection**
- First lookup step: enter keywords and find **peers responsible for these keywords** => obtain file hash
- Second lookup step: contact **peer responsible for the file hash** => obtain addresses of peers storing a copy of the file
- Generally, DHTs are well-suited to find **specific (and rare) data** efficiently
  - However, more inexact and approximate lookups are challenging



# DHT Refresher (5)

---

- Mechanism 3: **Direct downloads**
  - Although data is found in a multi-hop manner, download then takes place directly between two peers
- In systems with emphasis on anonymity (e.g., **Freenet**), this may be implemented differently: return path is also **multi-hop** (inefficient)
  - Peer does not know whether its neighbor requested the file or whether it is simply a forwarder



# Kad Network (1)

---

- The Kad network is the **most popular DHT** today
  - in fact, while DHTs have been a successful concept in literature, the predominant systems in practice are still server-based
  - Kad network consists of around **4 million peers**
  - about half of these peers can be **contacted directly** (no firewall or NAT)
  - it is based on the **Kademlia paper** by Maymounkov and Mazières
- Kademlia is also used in the **Overnet** p2p system
  - proprietary protocol, „shut down“ 2006

Much data from measurement studies of the Kad network as well as implementation details can be found in the recent papers by Steiner and Biersack.

# Kad Network (2)

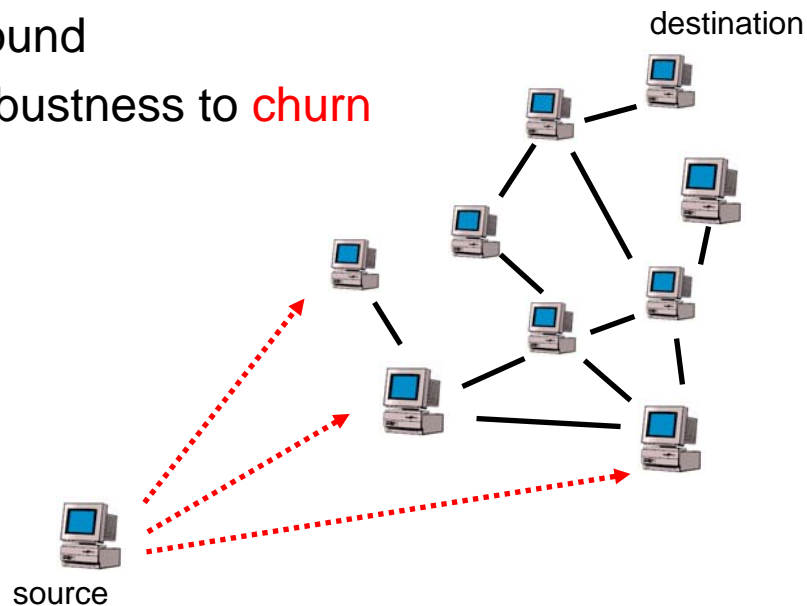
---

- Main concepts
  - each peer has **128-bit ID** (usually created by **random generator**)
  - ID defines position in cyclic ID space
  - **stored** at peer and reused when peer joins the network again
  - „hypercubic“ routing via **XOR metric**
  - for each  $i \in [0, 127]$ , a peer stores some contacts with distance between  $2^i$  and  $2^{i+1}$  to its own location
  - yields **logarithmic network diameter**
  - for each contact, peer stores: <Kad ID, IP address, port>
  - **replication policy** (typically 10 replicas in zone of peers which share first 8 bits)
  - 8-bit zone called „**tolerance zone**“, beneficial under **churn**
  - periodically **republished**
- In zone of **8-bit**, in one day, measurement studies observed **1.4 million publications** of files by 1.5 million distinct users and with 42,000 different keywords

# Kad Network (3)

- **Iterative routing**

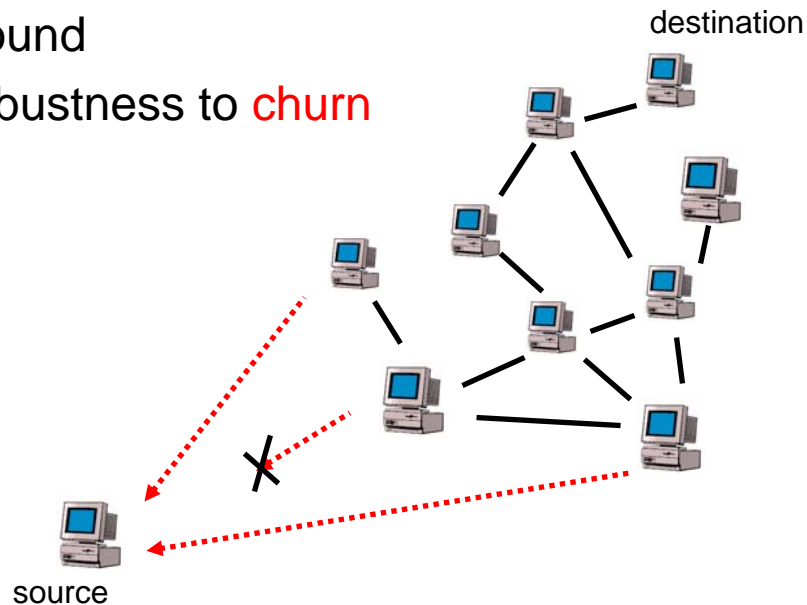
- in contrast to **recursive routing**
- requester runs **3 parallel lookups** which return new peers
- from them, requester selects 3 peers closer to destination
- and so on!
- termination: no closer peer found
- higher **delay** but improved robustness to **churn**



# Kad Network (3)

- **Iterative routing**

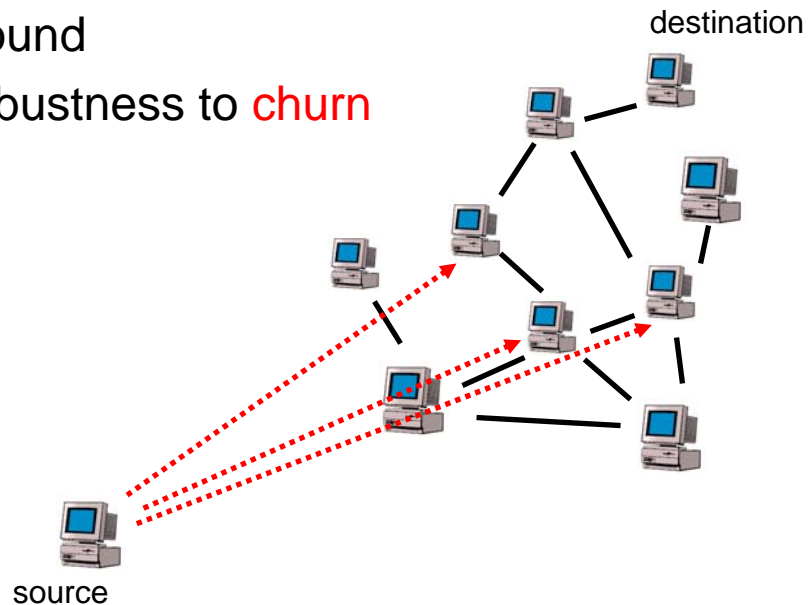
- in contrast to **recursive routing**
- requester runs **3 parallel lookups** which return new peers
- from them, requester selects 3 peers closer to destination
- and so on!
- termination: no closer peer found
- higher **delay** but improved robustness to **churn**



# Kad Network (3)

- **Iterative routing**

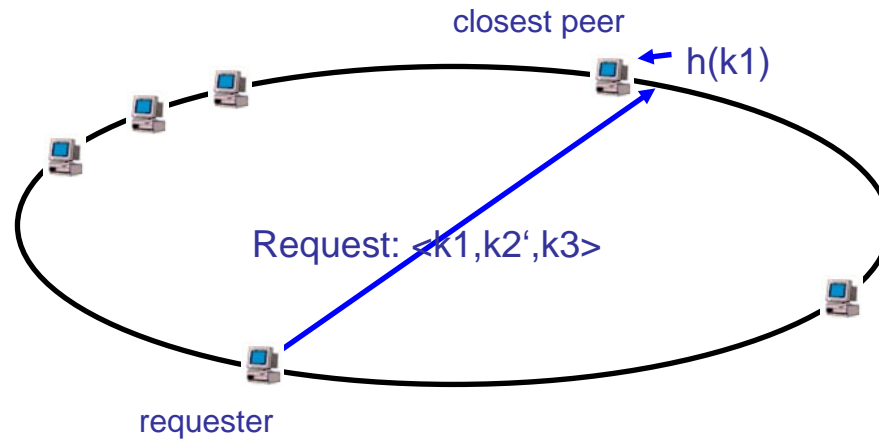
- in contrast to **recursive routing**
- requester runs **3 parallel lookups** which return new peers
- from them, requester selects 3 peers closer to destination
- and so on!
- termination: no closer peer found
- higher **delay** but improved robustness to **churn**





# Kad Keyword Request

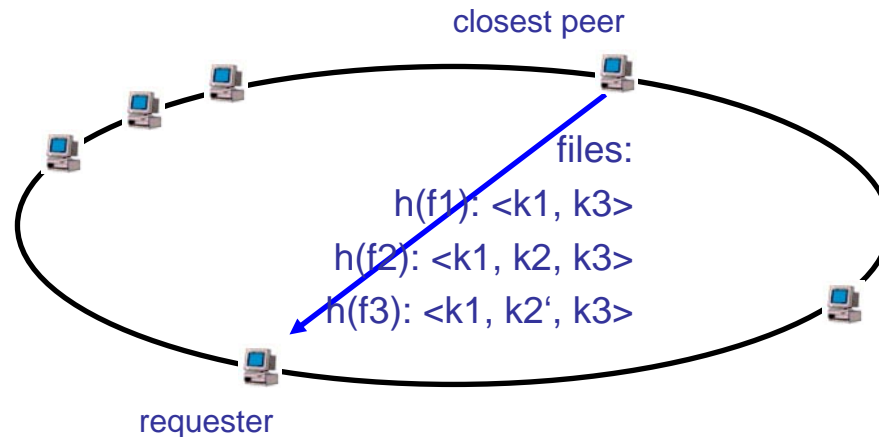
---



Lookup only with **first keyword** in list. Key is hash function on this keyword, will be routed to peer with Kad ID closest to this hash value.



# Kad Keyword Request

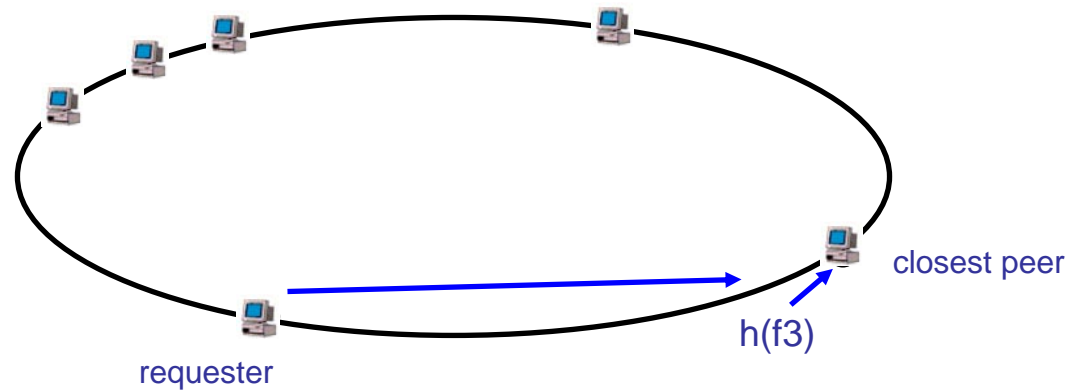


Peer **responsible** for this keyword returns different sources together with keywords.  
(remark: only those files with entries that include remaining keywords of request are returned, see later)



# Kad Source Request

---

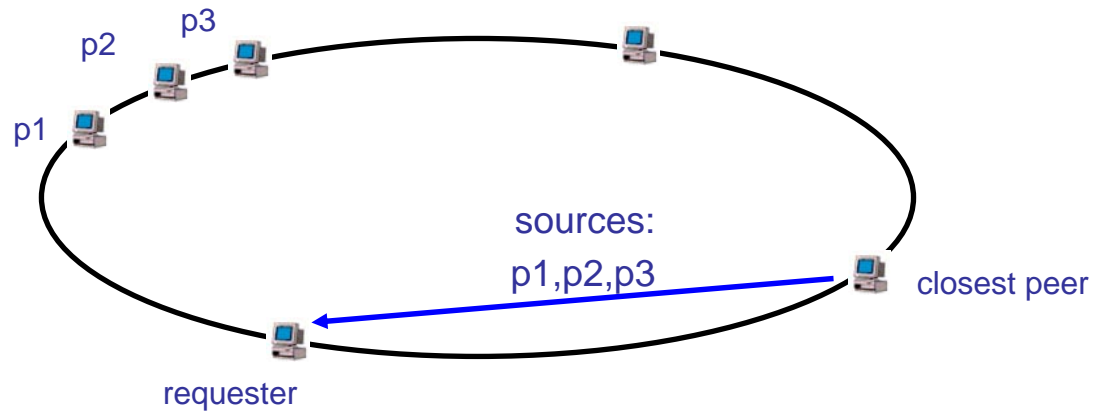


Peer can use this hash to find  
peer **responsible for the file**  
(possibly many with same content  
/ same hash)



# Kad Source Request

---

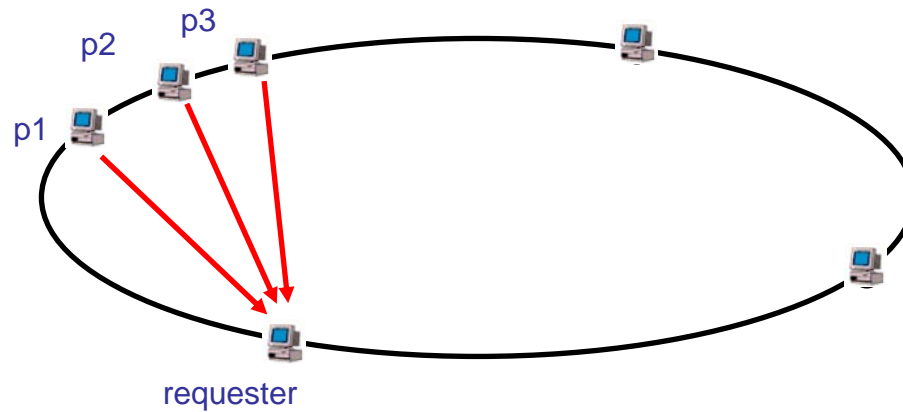


Peer provides requester with a list of peers storing a copy of the file.



# Kad Download

---

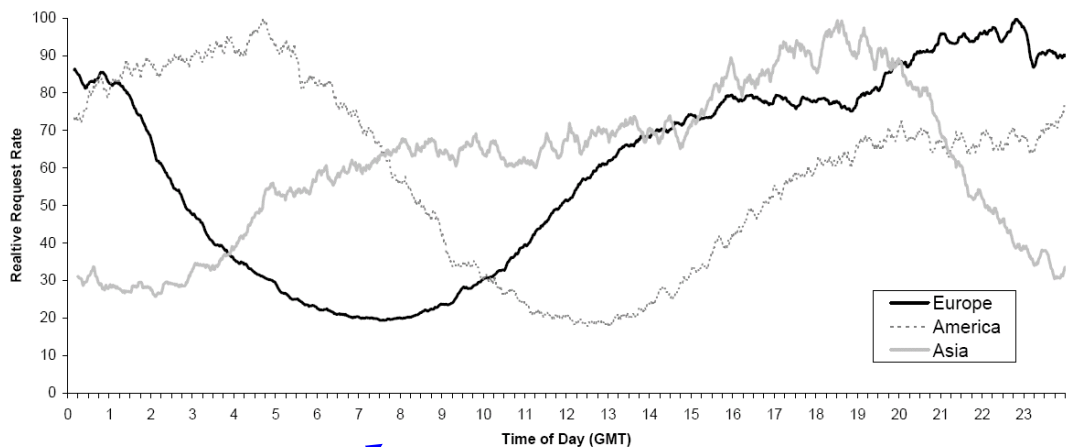


Eventually, the requester can download the data from these peers.



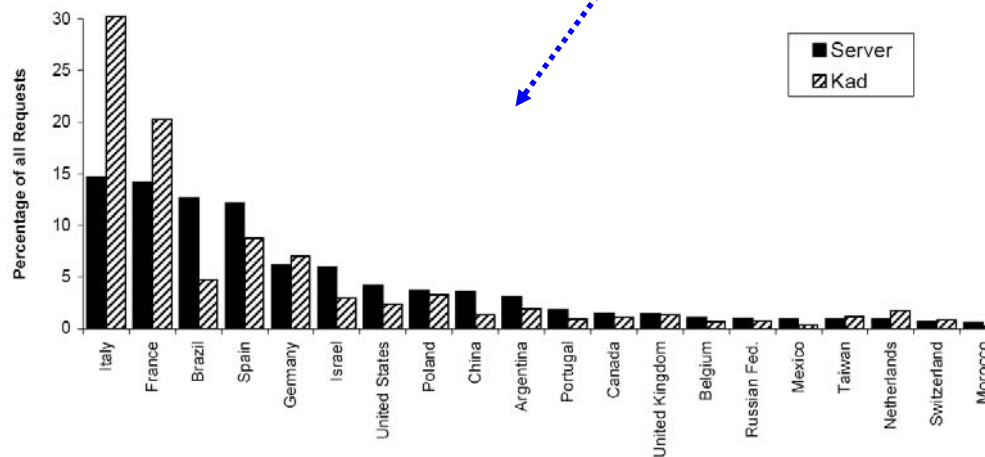
# Some Data

- In 2007, we received roughly **8 requests per minute** in Kad for the keyword „**Simpsons**“ (which also includes queries for „Simpsons Movie“, „Simpsons Sountrack“, etc.)



Quite popular in Europe (why this difference between eDonkey and Kad?)

Most Kad activity during evening / night (same for eDonkey)



# Some Challenges

---

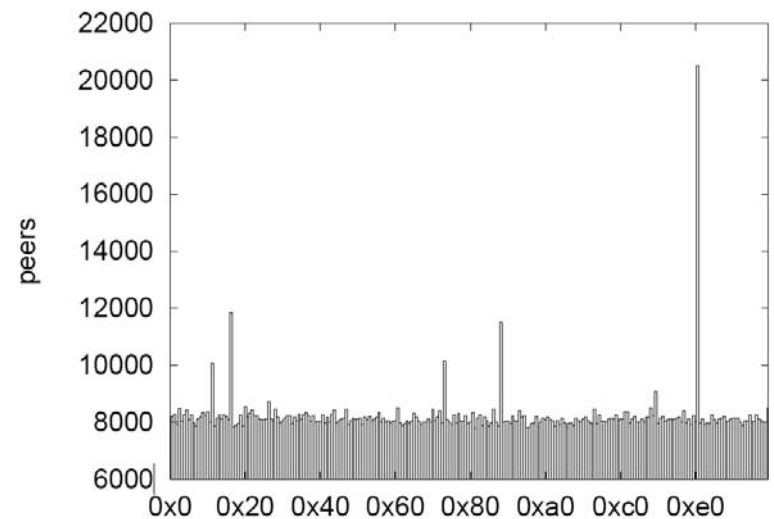
- Peer-to-peer principles also play a role in certain discussions about the design of a **future Internet**
  - e.g., to **disburden hotspots**
- Therefore, interesting to study today's **state-of-the-art** systems
- Some challenges that Kad currently faces...
  - case study: **ID assignment**



# Kad ID Assignment (1)

---

- Recall: each peer in Kad chooses a random ID
  - e.g., created with a **local random generator**
- Kad does not include any mechanisms to **verify** whether this ID has been produced „**properly**“
- Problem: choosing IDs can be used for attacks or for spying
  - indeed, many irregularities observed in today's Kad network
  - e.g., peers in China often change ID, non-uniform ID space, etc.
  - exploit can be used, e.g., for  **censorship**

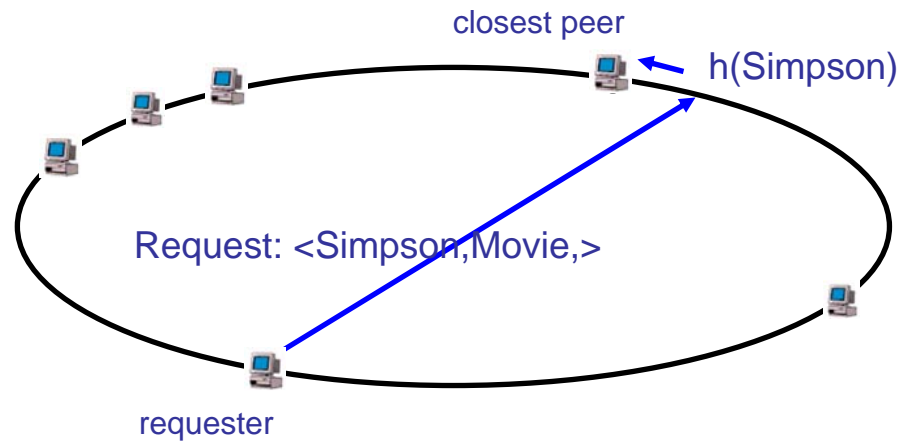




# Kad ID Assignment (2)

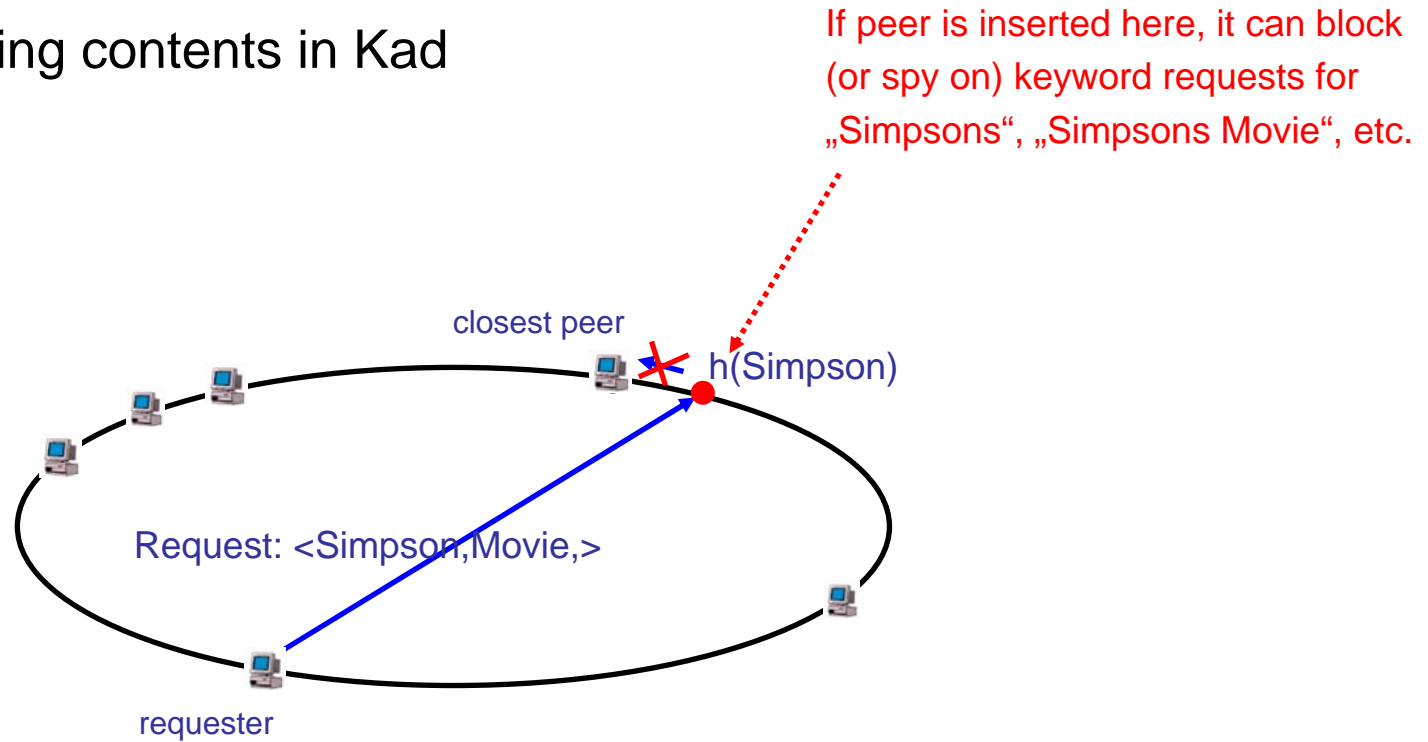
---

- E.g., censoring contents in Kad



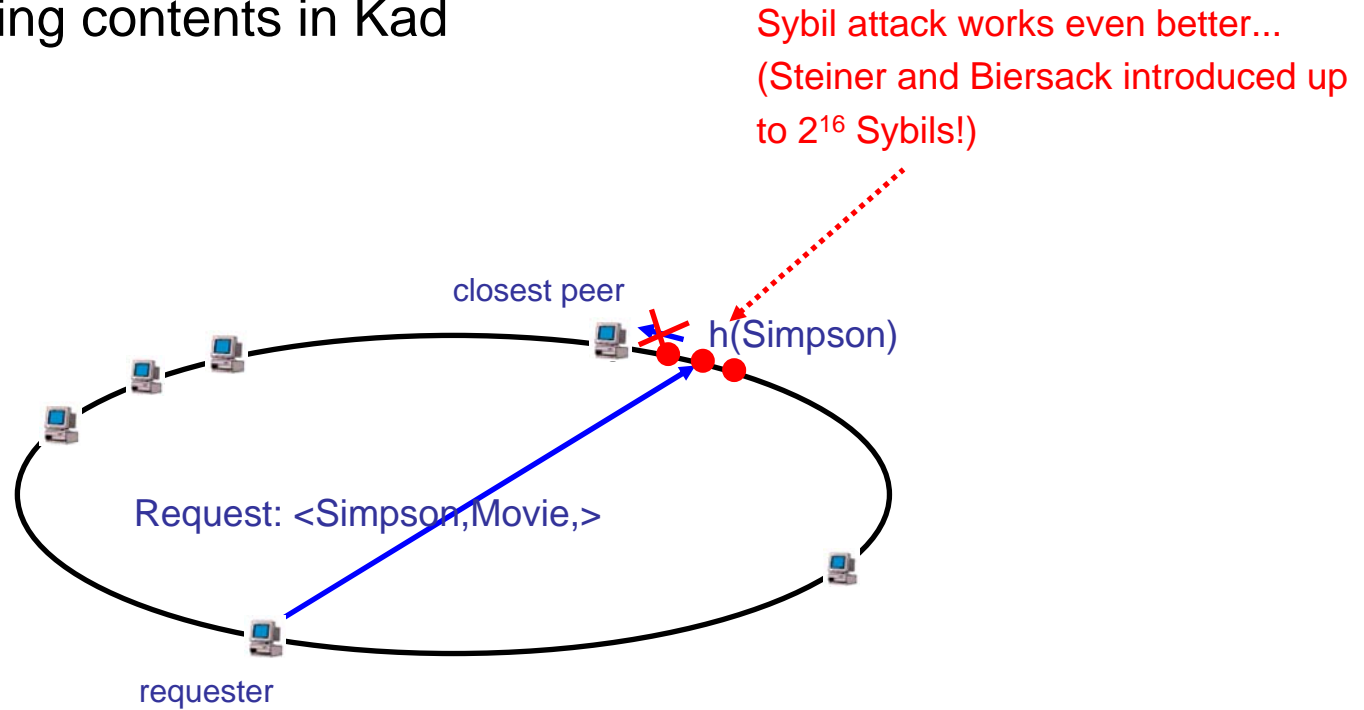
# Kad ID Assignment (2)

- E.g., censoring contents in Kad



# Kad ID Assignment (2)

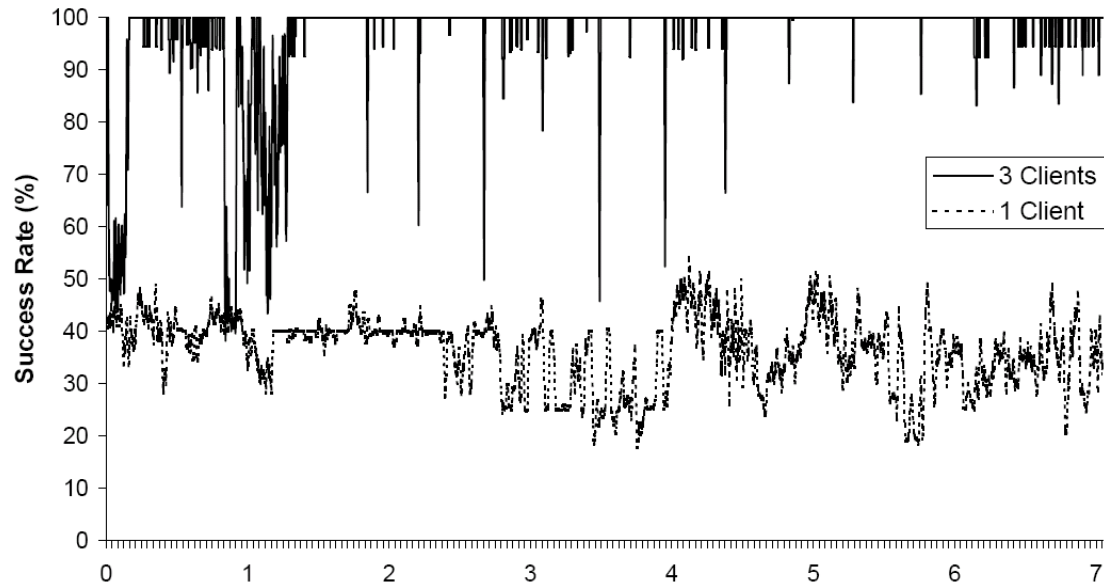
- E.g., censoring contents in Kad



# Kad ID Assignment (3)

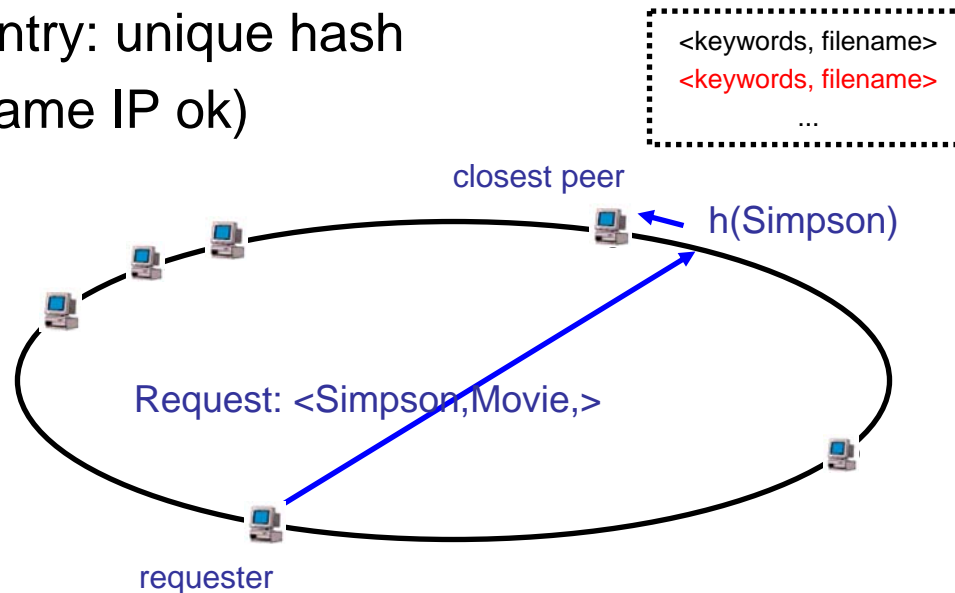
---

- Some data



# Additional Censorship: Publish Attack (1)

- Besides this „**peer insertion attack**“, additional censorship attacks exist
- For instance, a „**publish attack**“
  - We can also attack the originally publishing peers...
  - ... by creating **fake entries**
  - For each entry: unique hash (but from same IP ok)



## Publish Attack (2)

---

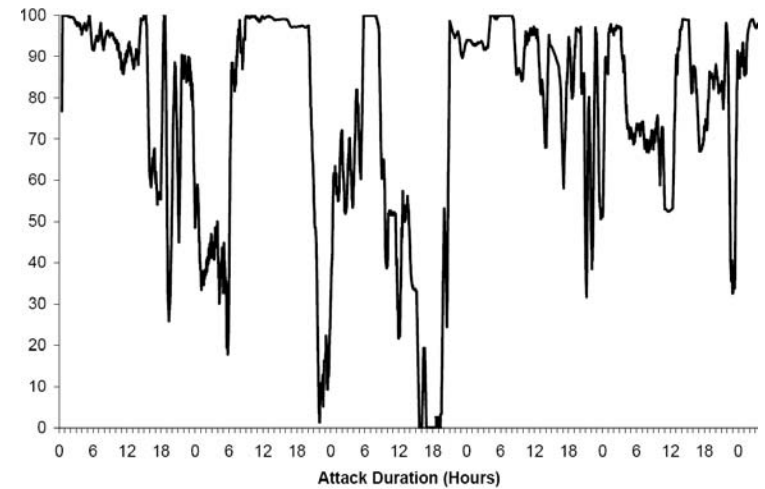
- Publishing peers return **at most 300 result tuples** per request
- Give priority to **latest additions** to index table
- Every entry expires after a couple of hours
- More difficult: attacked entry must include **superset of keywords** from the request
  - not known in advance: include **interpreter** name, label, etc.



# Publish Attack (3)

---

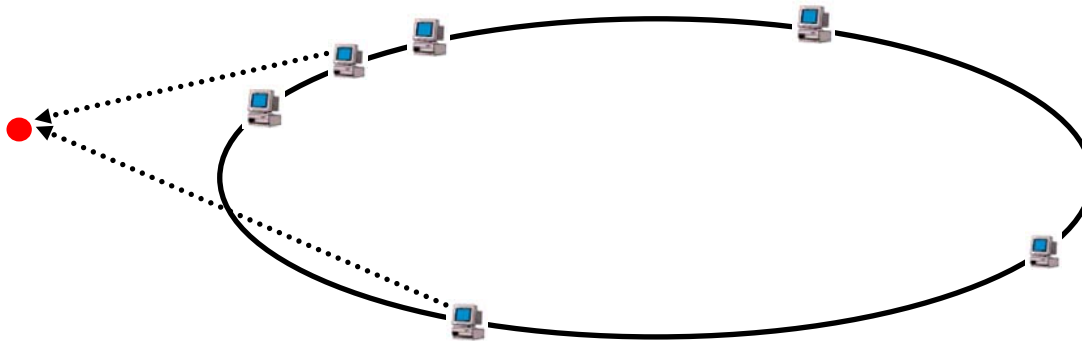
- Problem: Publishing peers accept many tuples from **same peer / IP address!**
- Less successful: some peers are **immune**



# Other Attacks

---

- It's possible to fill neighbor tables of peers
  - „eclipses“ this peer (**eclipse attack**)
- **DDoS attack**: publish attack can also be used to overwhelm peers outside the network with reuquests





# Countermeasures? (1)

---

- It seems that these attacks can easily be prevented
  - important insight: do not accept too much information from **same peer!**
  - do not allow peers to **choose their ID!**
- A solution? Choose overlay ID depending on IP address
  - e.g., a hash function on the IP address can be **verified!** (e.g., **Azureus**)
  - but what if IP address changes over time (**dynamic** IP addresses / DHCP)?
  - e.g., peers should not lose their credits when their IP changes
  - many peers have same IP address if behind a **NAT!**
  - other idea: compute a hash of **user-generated data** (e.g., a password) rather than of the IP address; thus, many different strings need to be tried to produce a **specific ID**
  - however, as there are much less than  $2^{128}$  peers in a network, an **approximate ID** will do the job for a peer insertion attack, and this can be computed efficiently
  - finally, an attacker may indeed have access to **many IP addresses** etc.



## Countermeasures? (2)

---

- What about **Sybil attacks**?
  - Same peer joins many times (with same or different IP address)
  - Difficult in decentralized environment?
  - Centralized solutions? Send SMS to obtain unique ID (hash from mobile phone number)? Solve CAPTCHA?
  - etc.

Many of these problems are not trivial  
in purely decentralized environments,  
and further research is needed!



A Glimpse at Two Other „Popular“ Applications:  
Peer-to-Peer Telephony with Skype  
and Botnets

# Skype (1)

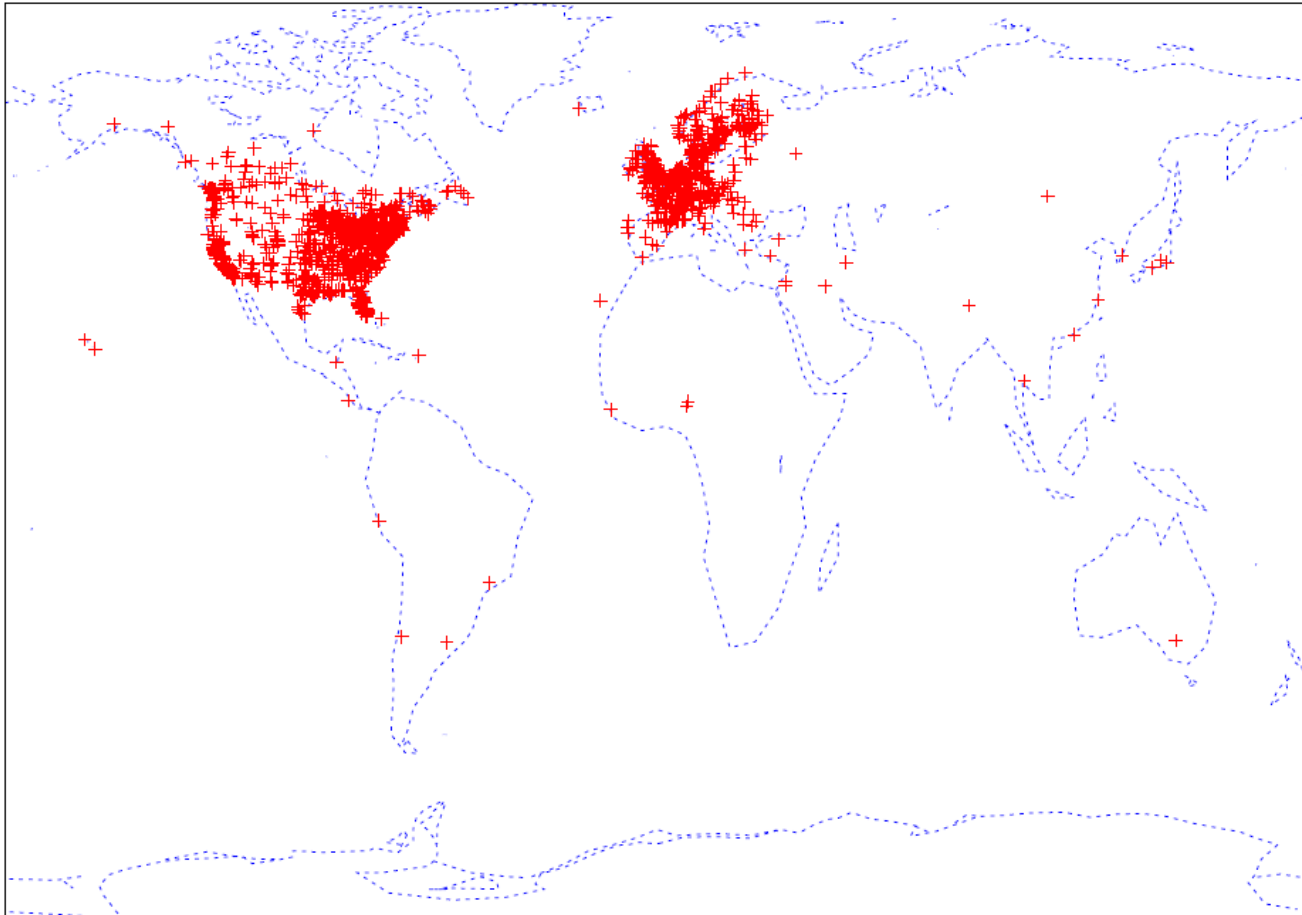
---

- Some facts...
  - VoIP network with more than **200 million users**
  - efforts to offer Skype on **mobile phones**, PSP, etc.
  - **proprietary** protocol, reverse-engineering difficult (many **papers** report on it... => ask me for details)
  - not interoperable with other VoIP networks
  - bought by **eBay** (for approx. 3.3 billion USD, October 2005)
  - according to Wikipedia: first quarter 2008 total of **14.2 billion minutes** skype-to-skype, 1.7 billion minutes skype-out, net revenue 126 million USD
- Predecessor file sharing system: **KaZaa** (FastTrack protocol)
  - two types of peers: **ordinary peers** and **super peers** (algorithms **unpublished**)
  - communication via **port 80** (problematic? spyware?)
  - super peer: public IP, sufficient CPU / bandwidth / memory / ...
  - UUHash algorithm used to allow downloading from **multiple sources** (checksum efficiently over parts of file)
  - but uploading only possible when entire file has been downloaded
  - UUHash algorithm problematic: **RIAA** used it to distribute fake files
  - no real incentive mechanism (**KaZaa lite** through reverse-engineering of ordinary peer – super peer communication...)

## Skype (2)

---

- Map of Skype supernodes (Xie&Yang, IPTPS 2007)



# Skype (3)

---

- Security
  - peer-to-peer calls **AES-256** encrypted
  - key exchange by 1536 or 2048 Bit RSA
- Traffic
  - phone call: approx. 30 MB per hour
  - however, background traffic up to **1 GB per month** (without any call)
  - traffic pattern can be problematic for ISPs (e.g., violating **no-valley routing policy** where customer relays traffic for its provider), claimed to increase costs



# Botnets (1)

---

- **Botnets** are one of the most significant threats in the **Internet** today
  - bot = program that performs tasks without user interaction
  - botnet = network of malicious bots that **illegally** control computing resources
  - some attackers are able to gain control of large portions of the Internet
- Used to disperse **spam**, conduct **DoS** attacks, etc.
- Keynote by Tom Leighton (Akamai) at **PODC 2007**:
  - 100s of servers under **DDoS attack** all the time
  - anti-virus company under constant attack since 2 years
  - some banks today pay **extortion money**
  - **4 large zombie armies** today, one tried to steal other three



## Botnets (2)

---

- Traditionally, botnets were coordinated **centrally**, e.g., via IRC chat
  - once identified, central IRC server can be taken down
- Now, first **peer-to-peer architectures** are emerging
  - e.g., **Peacomm**, aka Nuwar aka Zhelatin (= storm worm)
- E.g., paper by Grizzard et al. HotBots 2007





# The Peacomm Bot (1)

---

- Trojan.Peacomm botnet uses **Overnet** peer-to-peer protocol
  - i.e., Kademlia DHT
  - DHT provides communication primitive
  - allows peers to download secondary injections and to upgrade
- Protocol
  1. spread, e.g., via **email**
  2. connection to Overnet: initial list of peers **hard-coded** (bootstrap)
  3. download secondary injection (hard-coded keys to search and download an **encrypted URL**)
  4. hard-coded keys to decrypt URL
  5. download **secondary injection** from this URL
  6. execute



# The Peacomm Bot (2)

---

- Peer-to-peer protocol mainly used as a **name resolution server** for upgrading the bot
  - Peer-to-peer DNS with **encrypted data**
  - Data / URLs can change over time, nodes on which information is stored cannot be taken down (DHT...)
  - But keys indicate where data is (at least in ID space)
  - And bootstrap is also a weakness
- Secondary injections
  - e.g., to download additional components
  - e.g., SMTP **emailing** / spamming component
  - e.g., email propagation component
  - e.g., **DDoS** tool
  - etc.



# Conclusion



# Peer-to-Peer Backstage...

---

- Existing p2p systems are **heterogeneous** and dynamic
  - different goals (e.g., file sharing vs live streaming, anonymity, etc.)
- Some fundamental **concepts**
  - trend to structured p2p systems
- Interesting research **challenges**
  - incentive-compatibility
  - **robustness** to attacks
  - churn tolerance
  - in some sense, much research in distributed computing can be considered „peer-to-peer research“





Tack!

# Practical Issue: NATs and Firewalls

---

- **NAT** = network address translation
  - connection cannot be initiated **from outside** (no routing table entry)
- **Firewalls** can also be problematic for peer-to-peer systems
  - E.g., what if a peer in the eDonkey network is behind a firewall?
  - After client connected to server, server tried to contact client directly
  - If not possible, server assigns a so-called **lowID** to the client
  - If a peer  $p$  wants to download from a firewalled peer  $p'$  (having a lowID), the contact must be **mediated via the server**
  - Entails an additional overhead at server
  - Thus, highID clients can still download from lowID clients, however, **lowID-lowID download** remains impossible
- Similar techniques also work with Kad
  - An arbitrary „**buddy peers**“ assumes role of server
- Challenging topic, many more sophisticated solutions
  - e.g., clients such as NeoMule make lowID-lowID downloads possible
  - see also the **Skype protocol**...