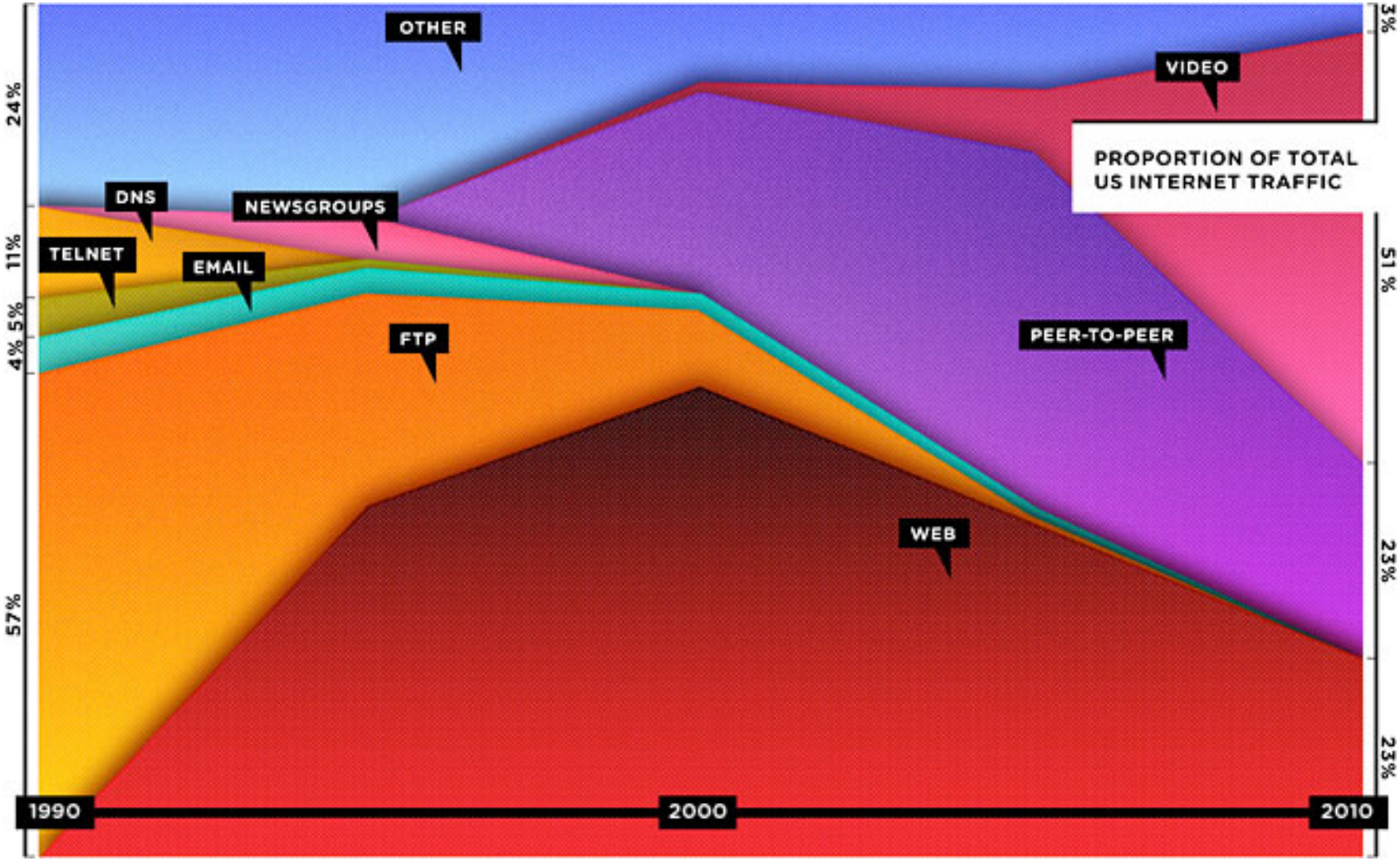
Peer-to-Peer Systems

Sistemas Distribuídos 2013/2014

Internet traffic



Categories

- **P2P file sharing**
  - Napster, Gnutella, KaZaA, eDonkey, BitTorrent, etc
- **P2P communication**
  - Instant messaging and Voice-over-IP: Skype
- **P2P computation**
  - SETI@home, PlanetLab
- **DHTs & their apps**
  - Chord, CAN, Pastry,Tapestry

Key Issues for P2P Systems

- **Join/leave**
  - How do nodes join/leave? Who is allowed?
- **Search and retrieval**
  - How to find content?
  - How are metadata indexes built, stored, distributed?
- **Content Distribution**
  - Where is content stored?
  - How is it downloaded and retrieved?

P2P Primitives

- **Join**
  - How to enter/leave the P2P system?
- **Publish**
  - How to advertise a file?
- **Search**
  - How to find a file?
- **Fetch**
  - How to download a file?

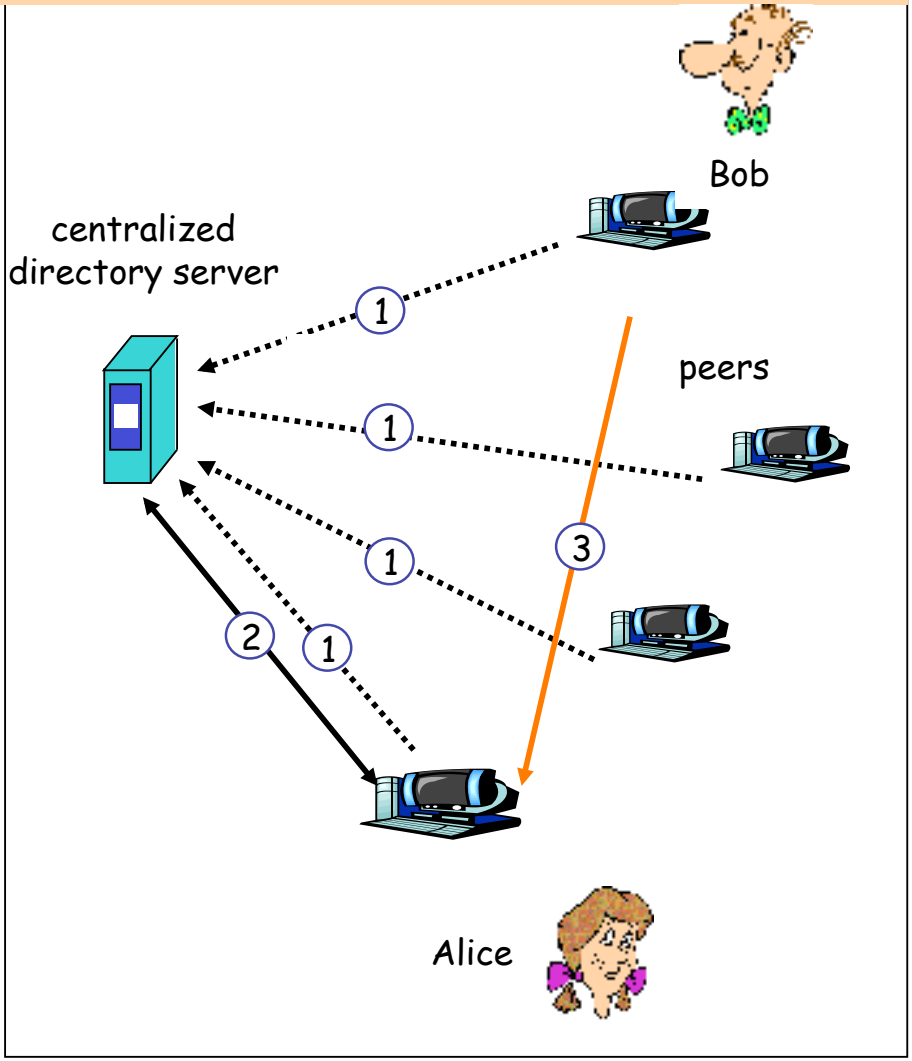
Publish and Search

- **Basic strategies:**
  - Centralized (Napster)
  - Flood the query (Gnutella)
  - Flood the index (PlanetP)
  - Route the query (Chord)

- Different trade-offs depending on application:
  - Robustness, scalability, legal issues

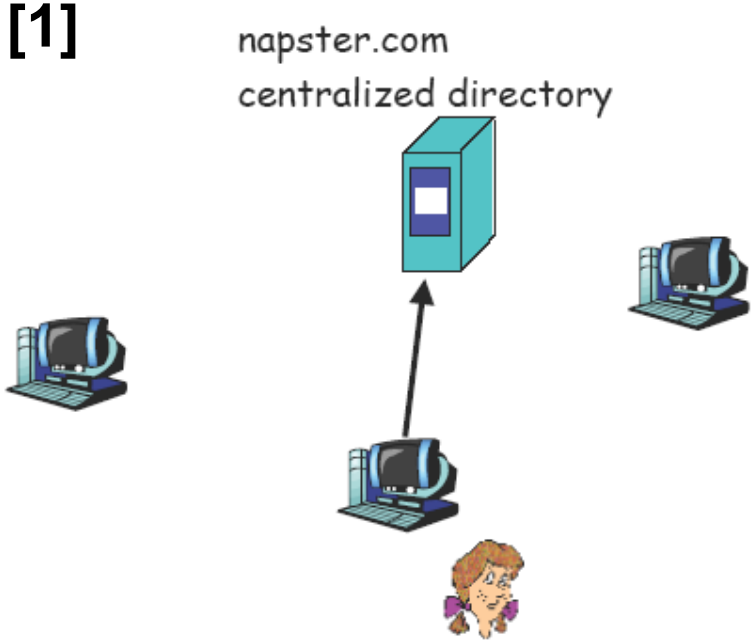
P2P: centralized directory

- **Centralized Database**:
  - **Join**: on startup, client contacts central server
  - **Publish**: reports list of files to central server
  - **Search**: query the server => return someone that stores the requested file
  - **Fetch**: get the file directly from peer

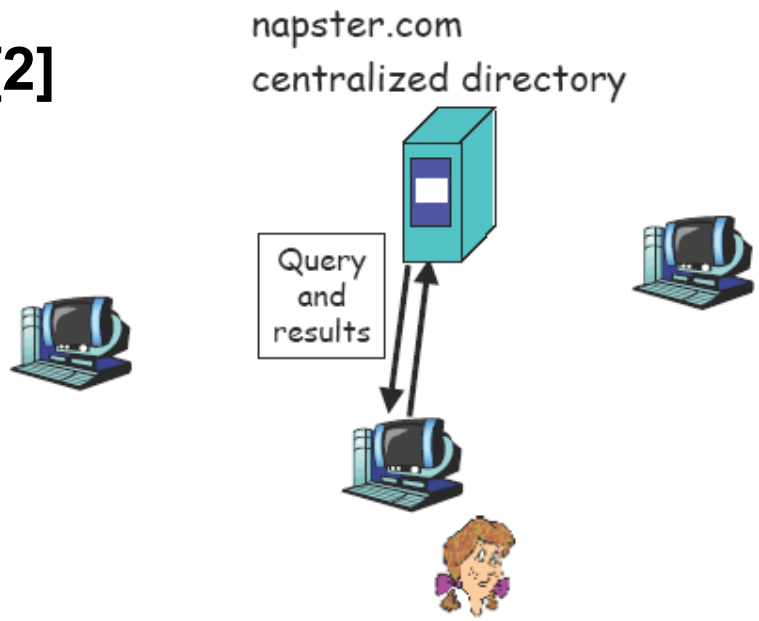


Napster

[1]

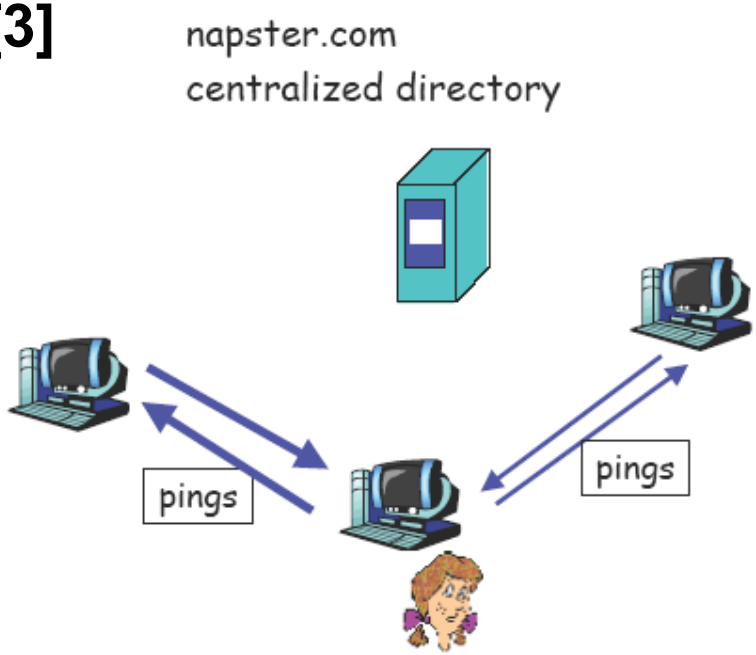

napster.com
centralized directory

[2]

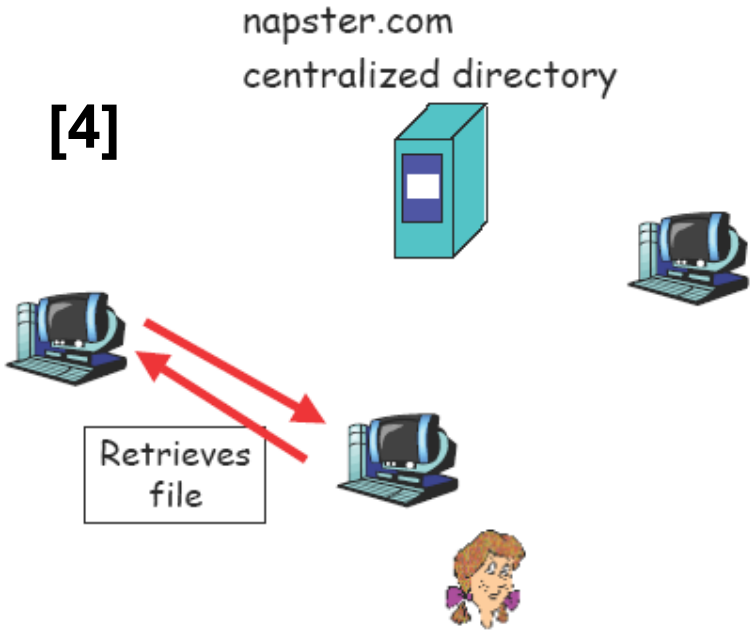

napster.com
centralized directory
Query
and
results

Napster

[3]


napster.com
centralized directory
pings
pings

[4]


napster.com
centralized directory
Retrieves
file

P2P: problems with centralized directory

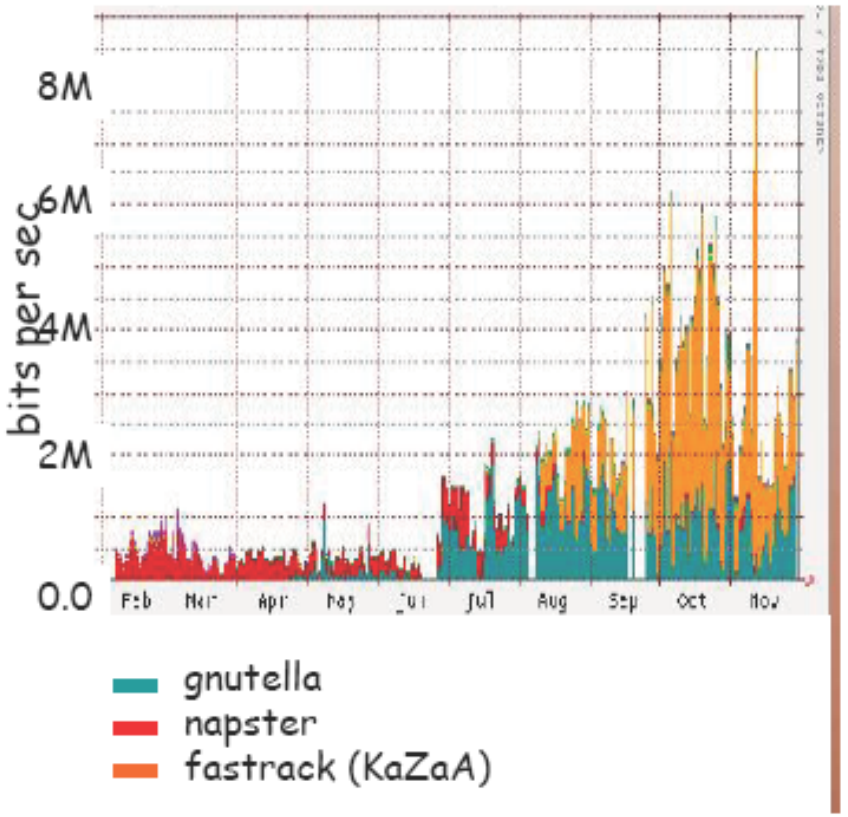
- Single point of failure
- Performance bottleneck (server maintains O(N) state)
- Copyright infringement

file transfer is decentralized,
but locating content is centralized

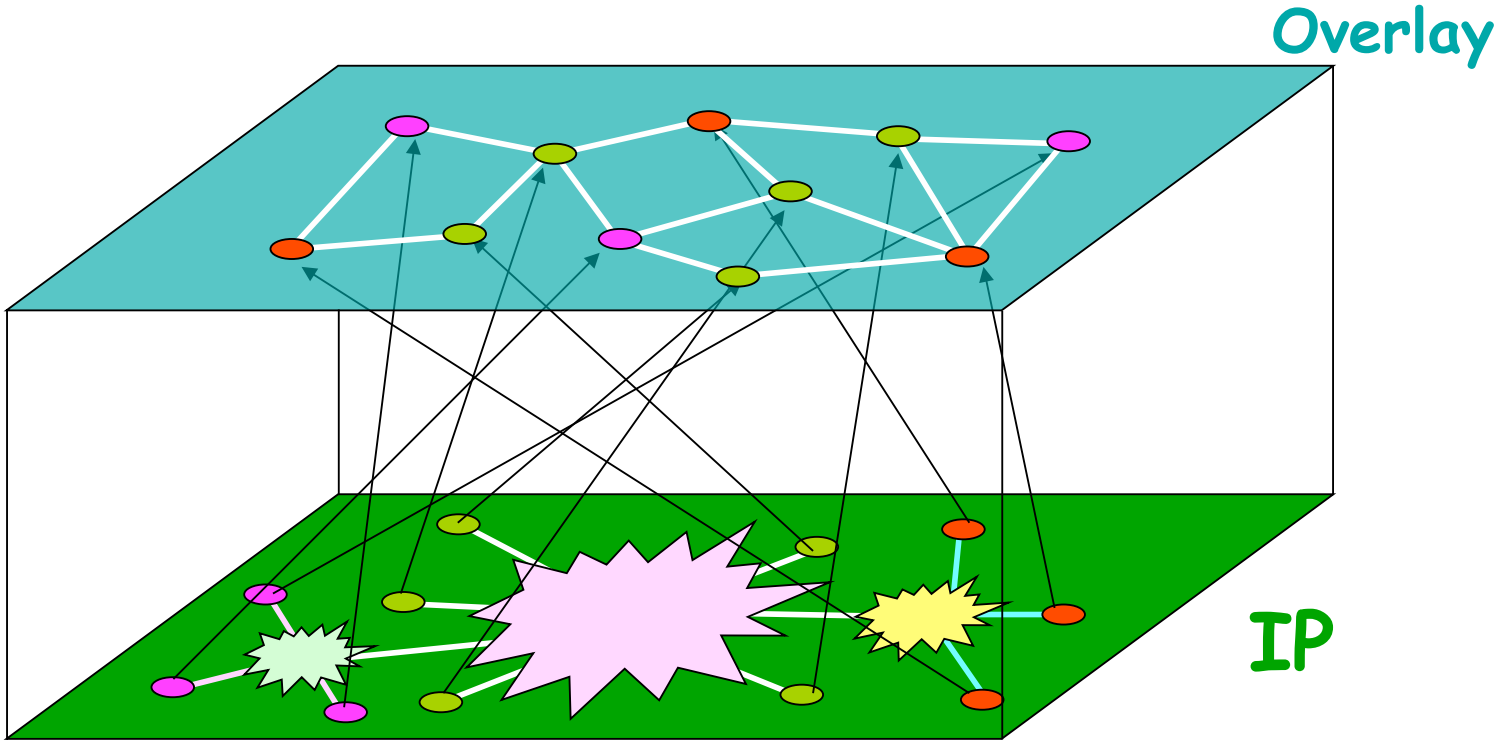
After Napster...

- In 1999, S. Fanning launches Napster
- Peaked at 1.5 million simultaneous users

- Napster was killed in 2001



Overlay Networks



Overlays

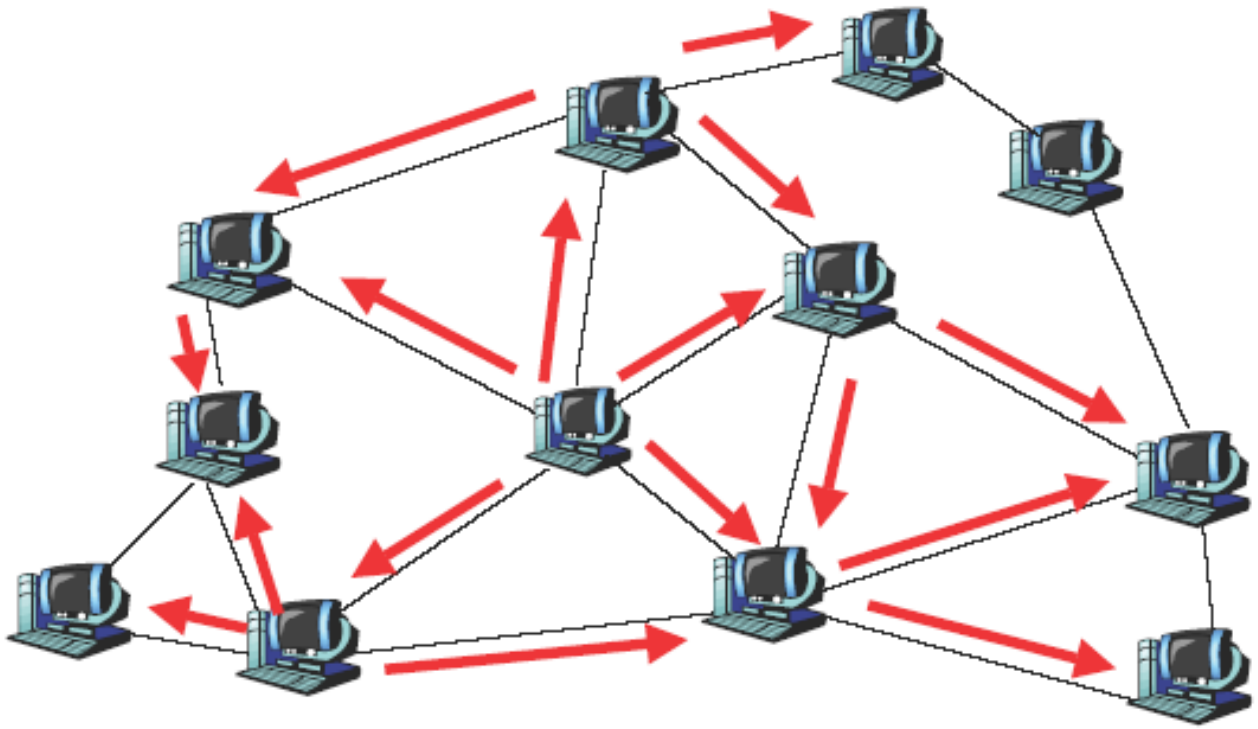
- **Unstructured overlays**
  - new node randomly chooses some existing nodes as neighbors
  - Examples: **Gnutella, KaZaA, BitTorrent, Skype**
- **Structured overlays**
  - peers arranged in a restrictive structure
  - Examples: **Pastry, CAN, Chord, Tapestry, ...**
- **Proximity**
  - Not necessarily taken into account

Unstructured P2P

Gnutella

- In 2000, J. Frankel and T. Pepper from Nullsoft released Gnutella
- Soon many other clients: Bearshare, Morpheus, LimeWire, etc.
- In 2001, many protocol enhancements including “ultrapeers”

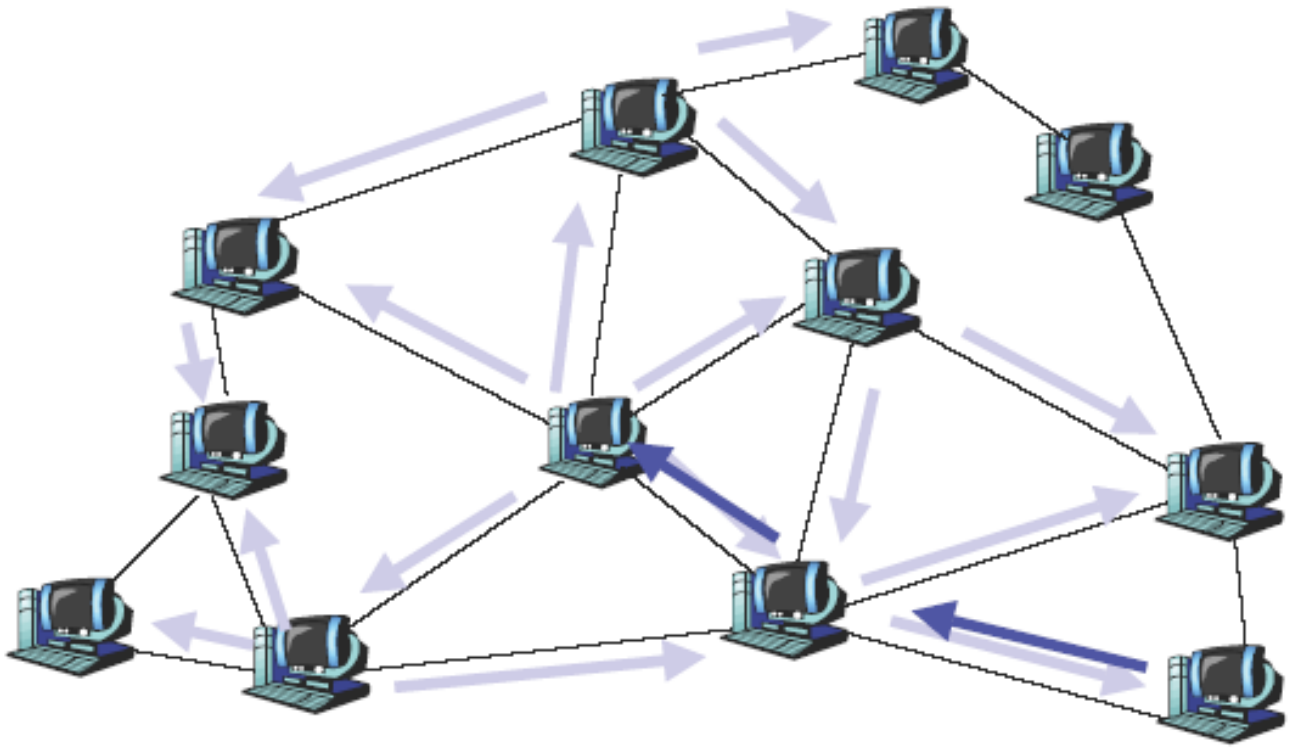
Gnutella: Unstructured P2P File Sharing



Gnutella

- Query Flooding:
  - **Join**: on startup, client contacts a few other nodes; these become its “neighbors”
  - **Publish**: no need
  - **Search**: ask neighbors, who as their neighbors, and so on... when/if found, reply to sender.
  - **Fetch**: get the file directly from peer

Gnutella: query flooding



Gnutella

- Completely decentralized
- More difficult to “pull the plug”
- More scalable?....

- Each application instance is used to:
  - store selected files
  - route queries from and to its neighboring peers
  - respond to queries if file stored locally
  - serve files

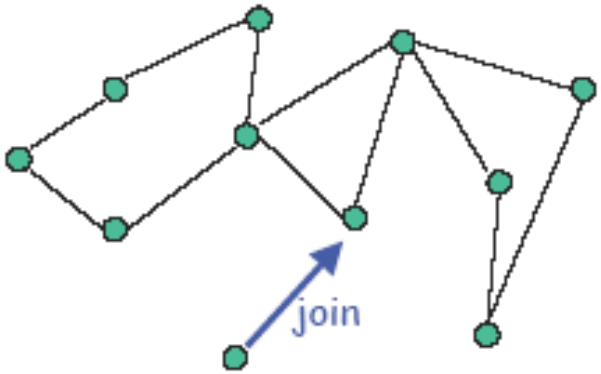
Querying...

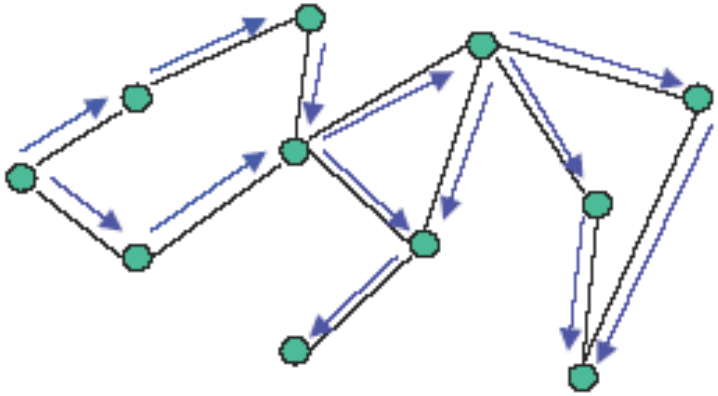
- Searching by flooding:
  - if you don't have the file you want, query 7 of your neighbors.
  - if they don't have it, they contact 7 of their neighbors, for a maximum hop count of 10
  - reverse path forwarding for responses

- This floods the networks with `QUERY` messages...

Gnutella: Overlay Management

- New node uses bootstrap node to get IP addresses of existing Gnutella nodes
- New node establishes neighboring relations by sending join messages





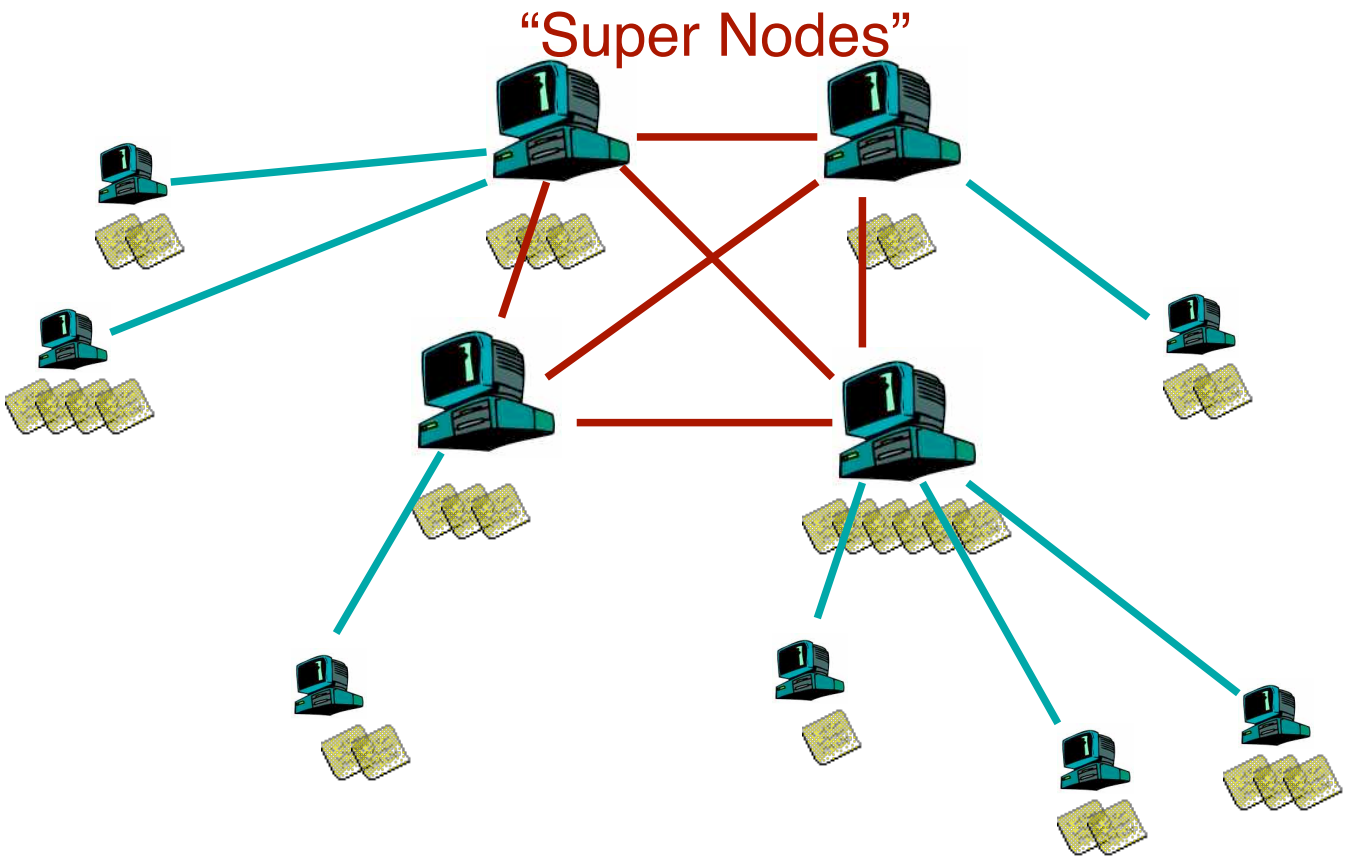
KaZaA

- In 2001, KaZaA created by Dutch company Kazaa BV
- Single network called FastTrack used by other clients as well: Morpheus, giFT, etc.

- More than 3 million up peers sharing over 3,000 Tbytes
- More popular than Napster

- Optional parallel downloading of files
- Automatically switches to new download server when current server becomes unavailable

Kazaa

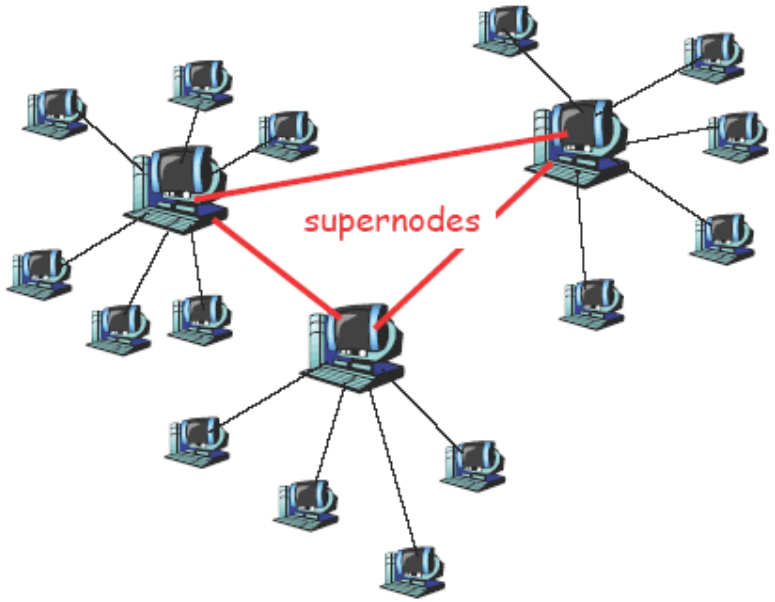


KaZaA

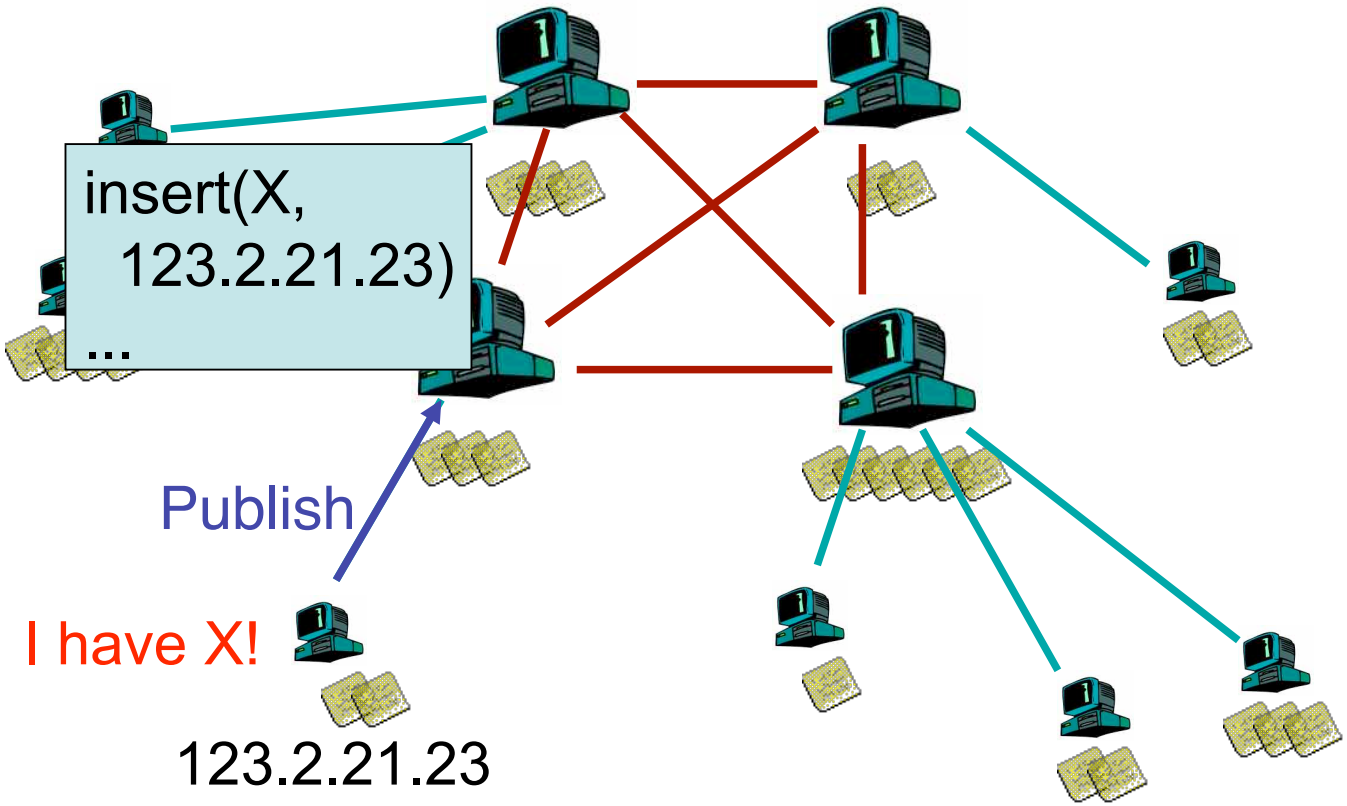
- “Smart” Query Flooding:
  - **Join**: on startup, client contacts a “supernode” ... may at some point become one itself
  - **Publish**: send list of files to supernode
  - **Search**: send query to supernode, supernodes flood query amongst themselves.
  - **Fetch**: get the file directly from peer(s); can fetch simultaneously from multiple peers

KaZaA: Architecture

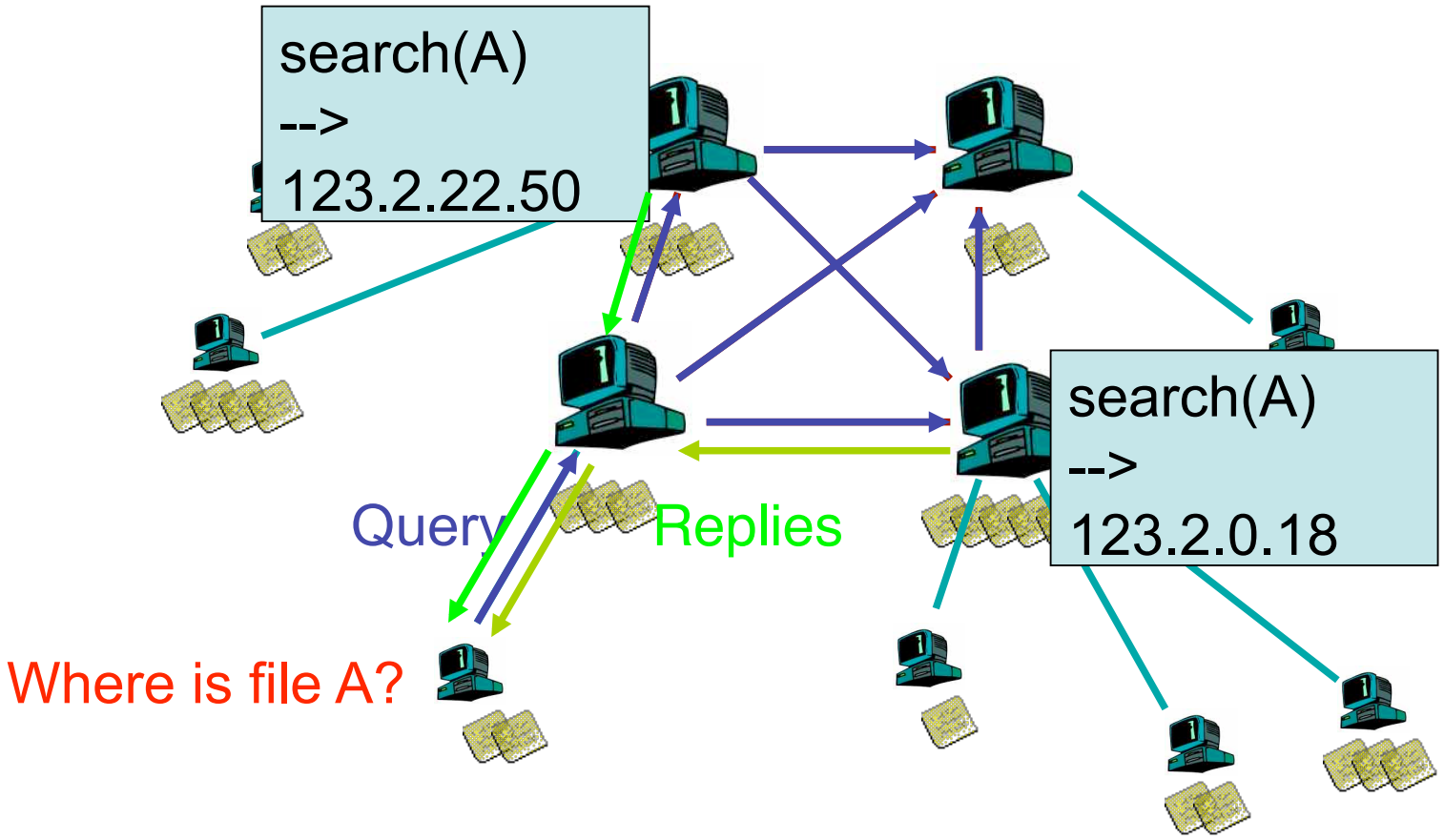
- Each peer is either a supernode or is assigned to a supernode
  - Each SN has about 100-150 children
  - Roughly 30,000 SNs
- Each supernode has TCP connections with 30-50 supernodes
  - 0.1% connectivity



Kazaa: file insert



Kazaa: file search



Kazaa: fetching a file

- More than one node may have requested the file...
- How to tell?
  - Must be able to distinguish identical files
  - Not necessarily same filename
  - Same filename not necessarily same file...
- Use Hash of file
  - KaZaA uses UUHash: fast, but not secure
  - Alternatives: MD5, SHA-1
- How to fetch?
  - Get bytes [0..1000] from A, [1001...2000] from B
  - Alternative: Erasure Codes

KaZaA: who are they?

- Software developed by Estonians
- FastTrack originally incorporated in Amsterdam
- FastTrack deploys KaZaA service

- Summer 2001, Sharman networks, founded in Vanuatu (small island in Pacific), acquires FastTrack
  - Board of directors, investors: secret

- Employees spread around, hard to locate

Skype Founders

- Niklas Zennstrom

- Janus Friis

AP / Sergio Dionisio

http://mashable.com/2011/05/10/microsoft-acquires-skype/ **(8.5billion)**

Bit-Torrent

- In 2002, B. Cohen debuted BitTorrent

- Key Motivation:
  - Popularity exhibits temporal locality (Flash Crowds)

- Focused on Efficient *Fetching*, not *Searching*:
  - Distribute the *same* file to all peers
  - Single publisher, multiple downloaders

- Has some “real” publishers:
  - Linux distributions
  - Software patches
  - Blizzard Entertainment (world of warcraft)

- Reference for more info:
  http://en.wikipedia.org/wiki/BitTorrent

Bit-Torrent: .torrent file

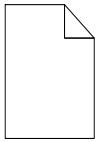
For each new BitTorrent file, one server hosts the original copy

- The file is broken into chunks

There is also a .torrent file which is typically kept on some web server(s)

.torrent file: Static 'metainfo' file that contains:

Name
Size
Checksum
IP address of the Tracker

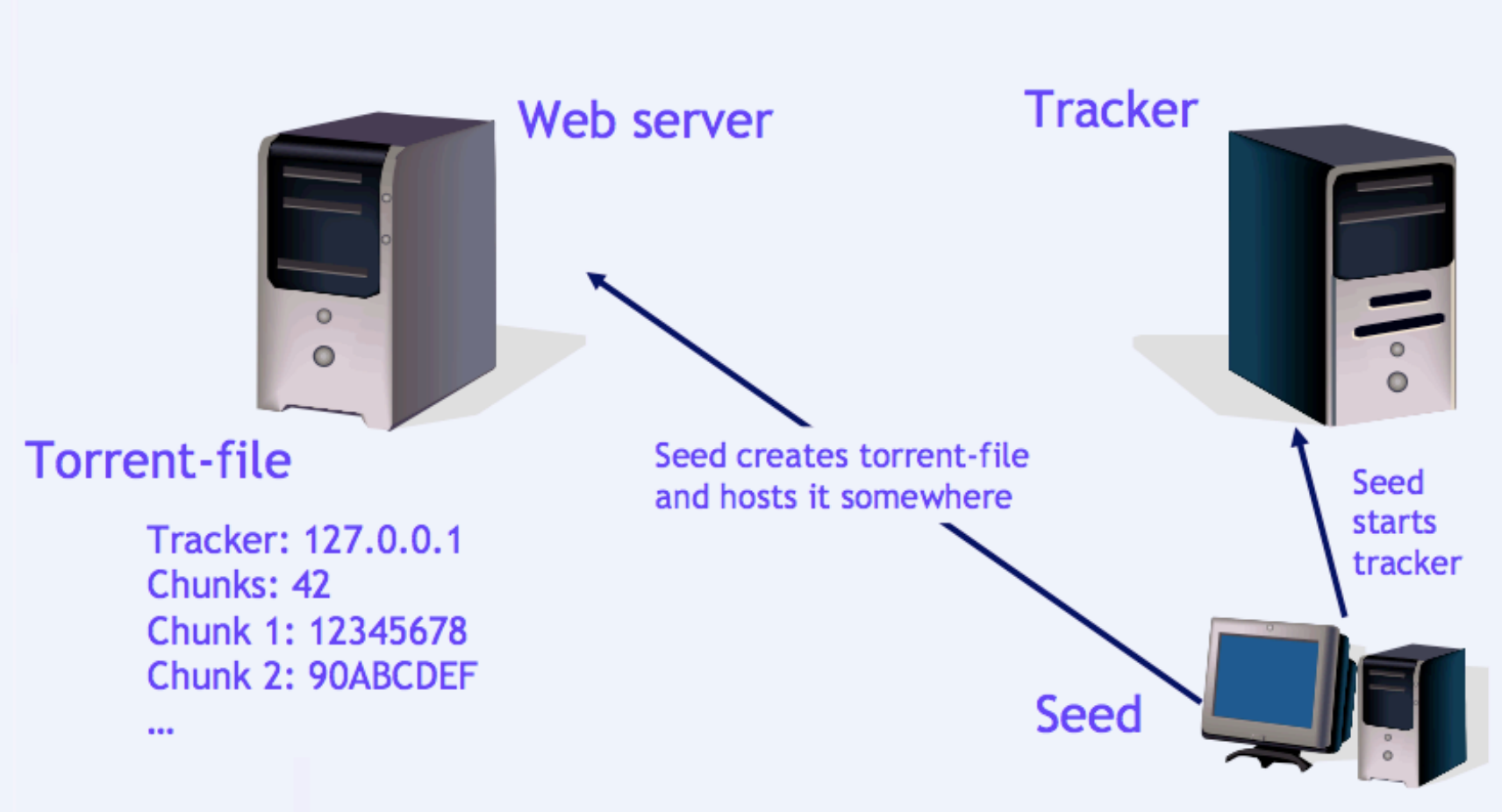


Matrix.torrent

Bit-Torrent: the Tracker

- The tracker is a separate server :
  - Keeps track of currently active clients for a file
  - Does not participate in the download
  - Never holds any data
  - However, some lawsuits have been successful against people running trackers!

Bit-Torrent: the seeder



Bit-Torrent: seeders and leechers

Terminology:

- Seeder: Client with a complete copy of the file
- Leecher: Client still downloading the file

Client contacts tracker and gets a list of other clients

- Gets list of 50 peers

Client maintains connections to 20-40 peers

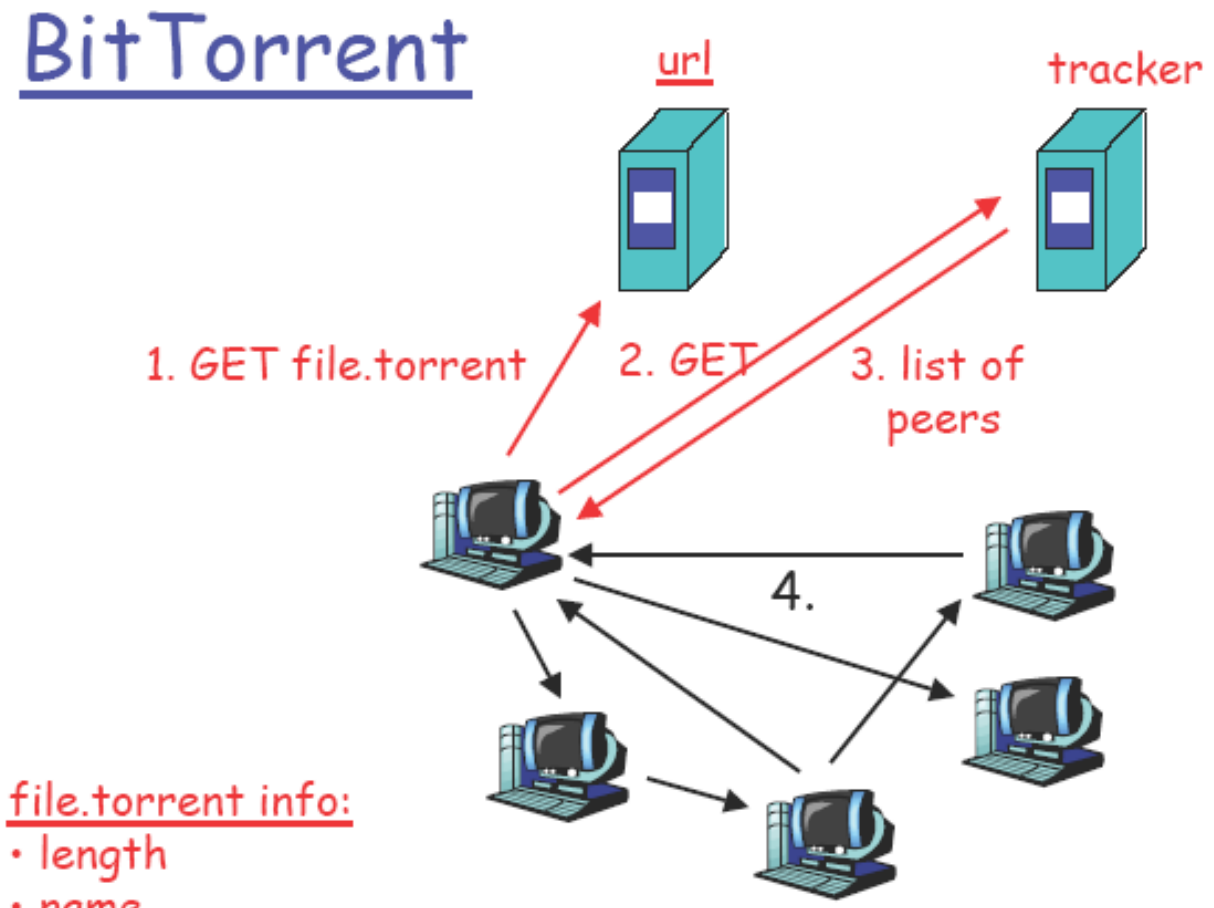
- Contacts tracker if number of connections drops below 20
- This set of peers is called peer set

Client downloads chunks from peers in peer set and provides them with its own chunks

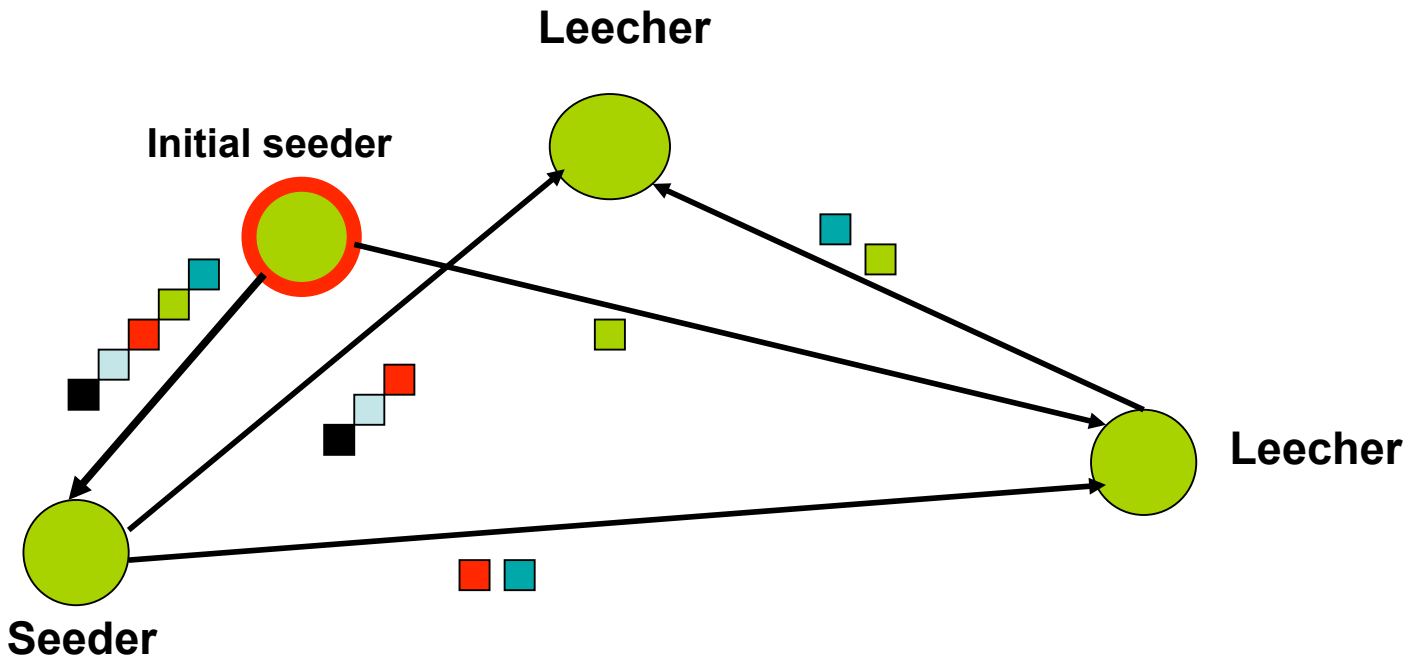
Chunks typically 256 KB

- Chunks make it possible to use parallel download

BitTorrent


BitTorrent
url
tracker
1. GET file.torrent
2. GET
3. list of peers
4.
file.torrent info:
• length
• name
• hash
• url of tracker

Seeders and Leechers



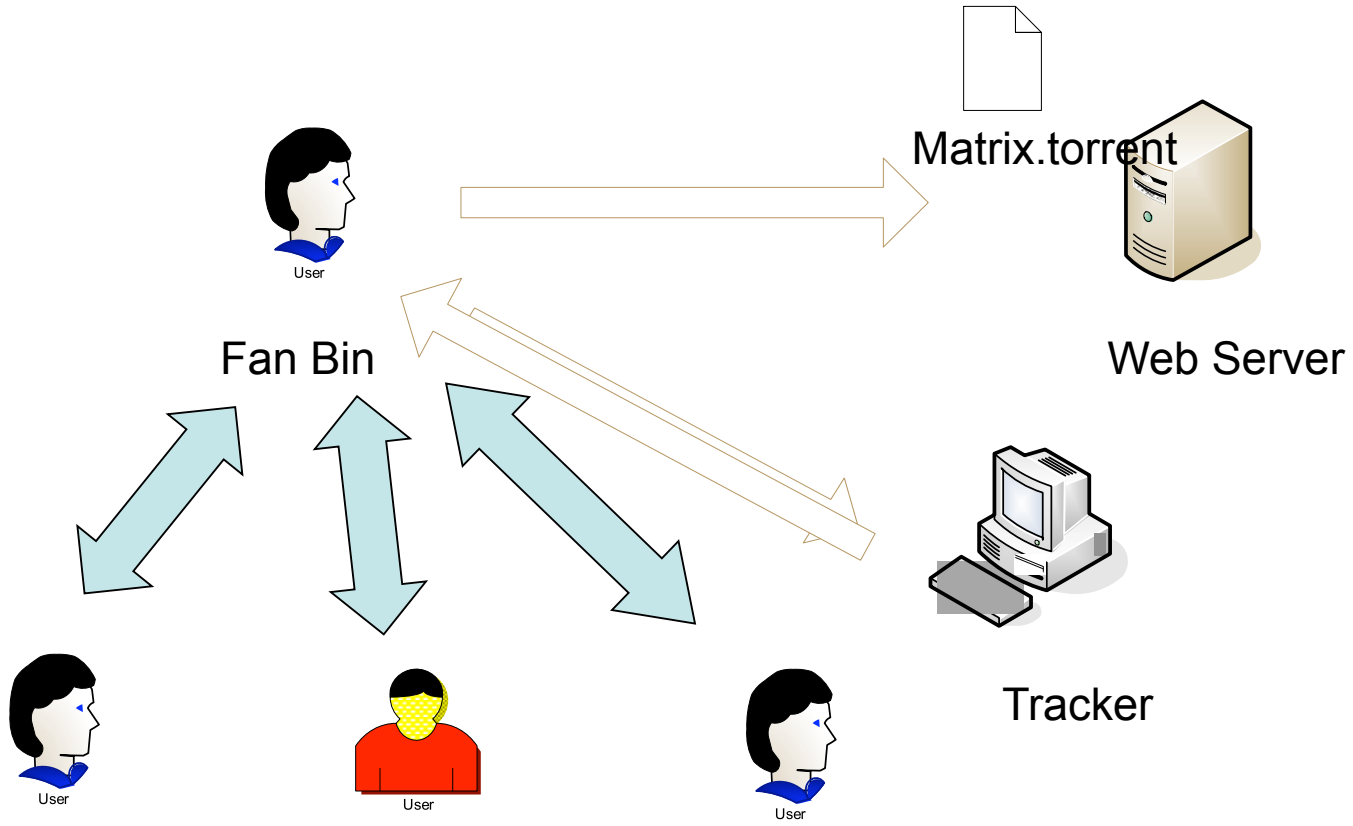
BitTorrent: overview

- Swarming:
  - **Join**: contact centralized “tracker” server, get a list of peers.
  - **Publish**: Run a tracker server.
  - **Search**: Out-of-band. E.g., use Google to find a tracker for the file you want.
  - **Fetch**: Download chunks of the file from your peers. Upload chunks you have to them.

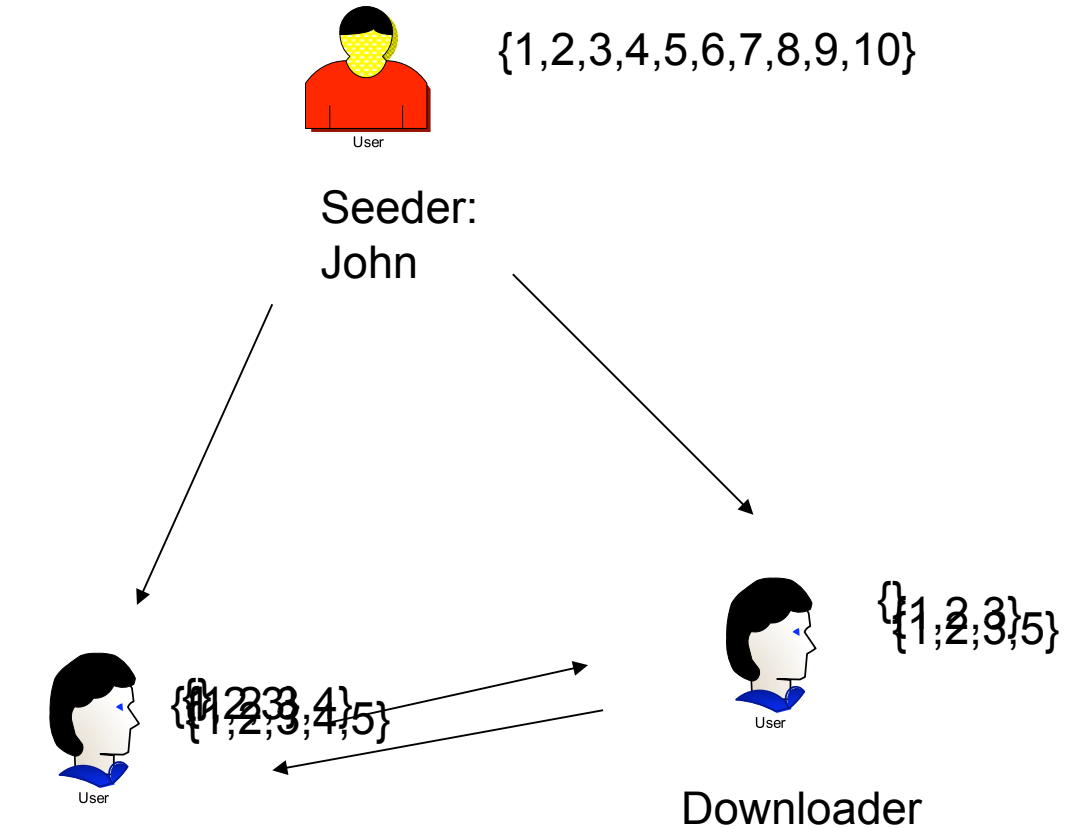
BitTorrent: tit-for-tat

- "**Tit-for-tat**" (encourage fair trading)
  - Bit-torrent uploads to at most 4 peers
  - Among the uploaders, upload to the 4 that are downloading to you at the highest rates.
  - This encourages some fair trading and tries to avoid free-riding.
  - Upload to nodes that have been also uploading.

BitTorrent: how it works

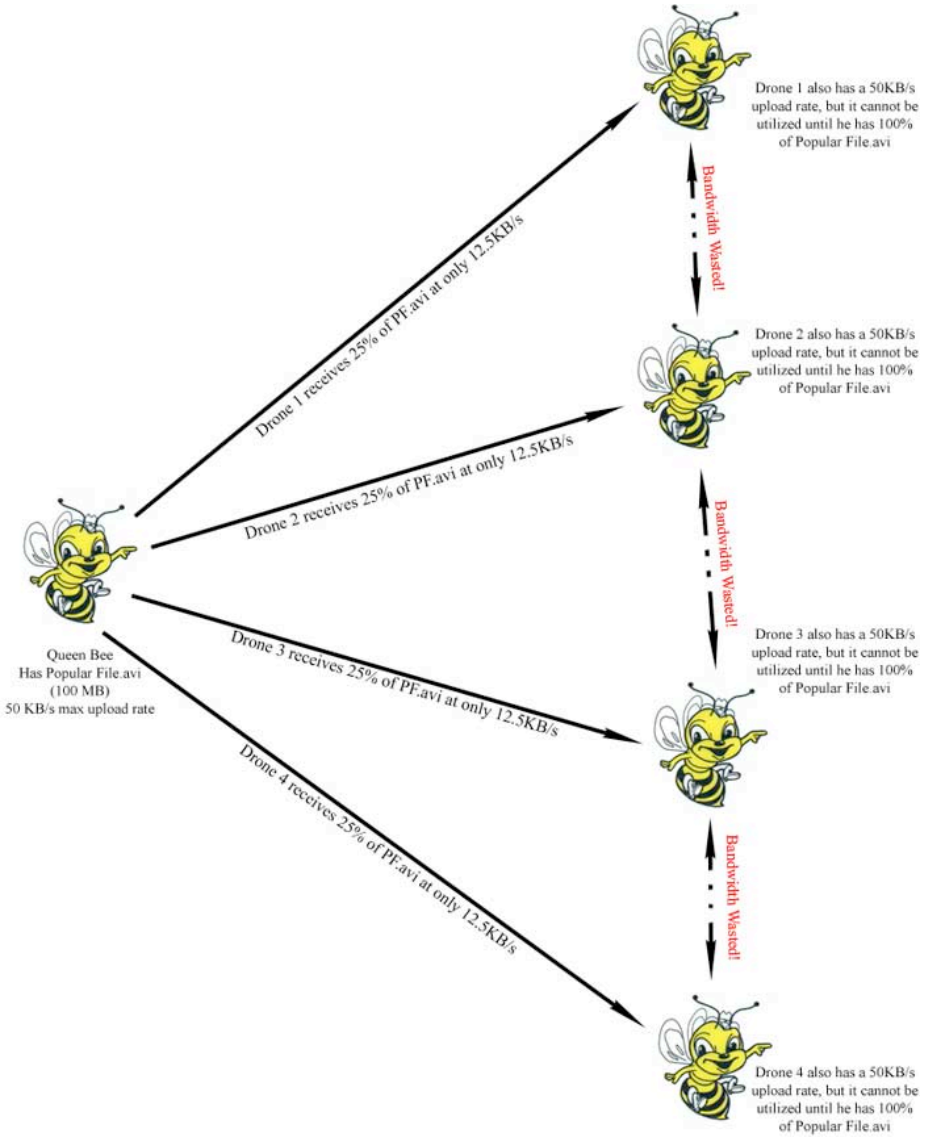


BitTorrent: swarming

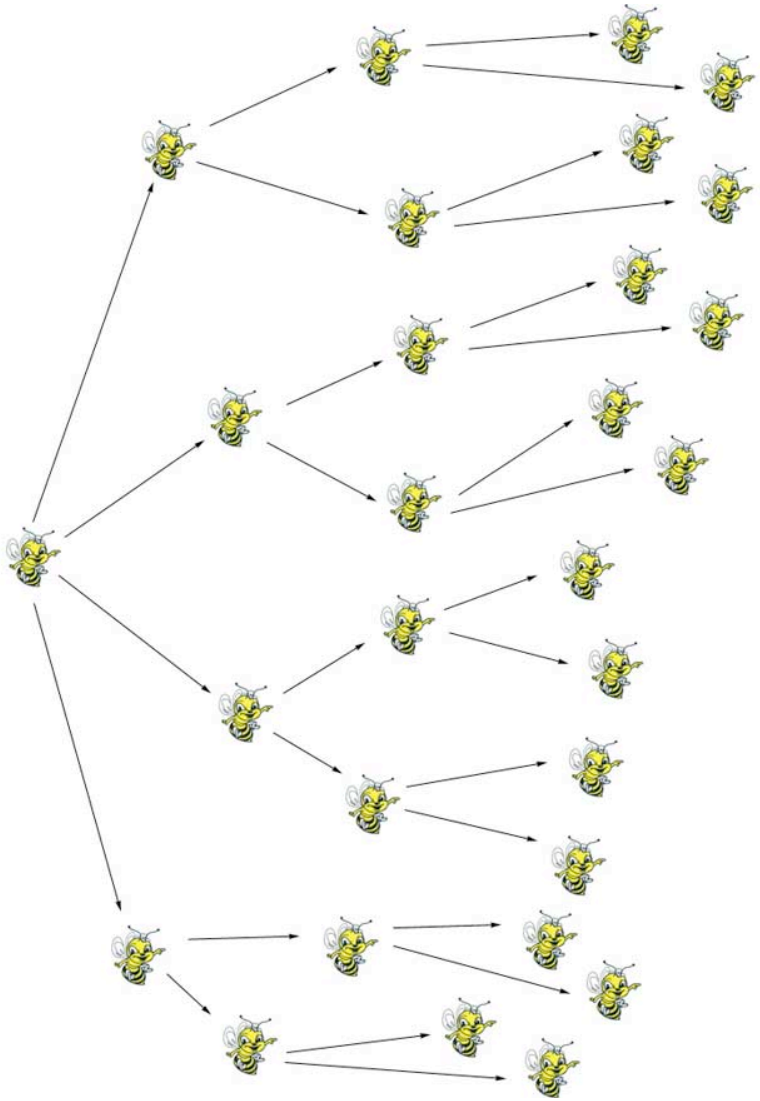


- To verify data, Hash codes are used for all the pieces, included in the .torrent file.

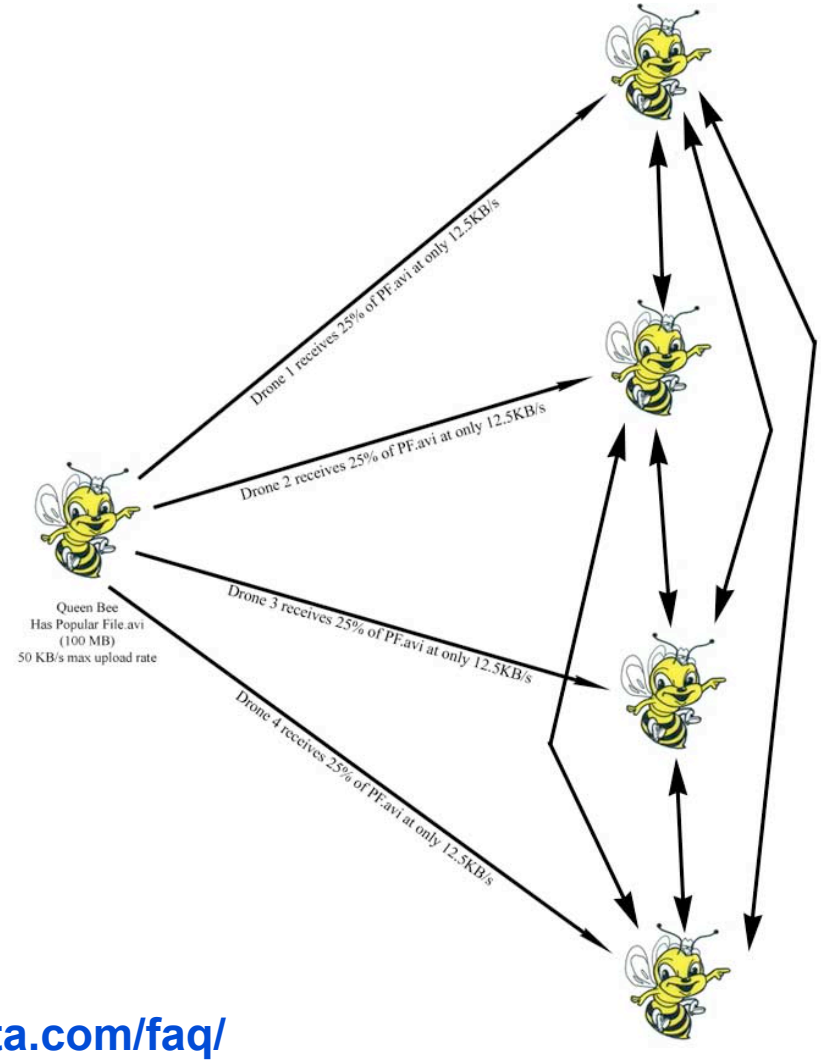
Example: Kazaa and similar



... after some popular download

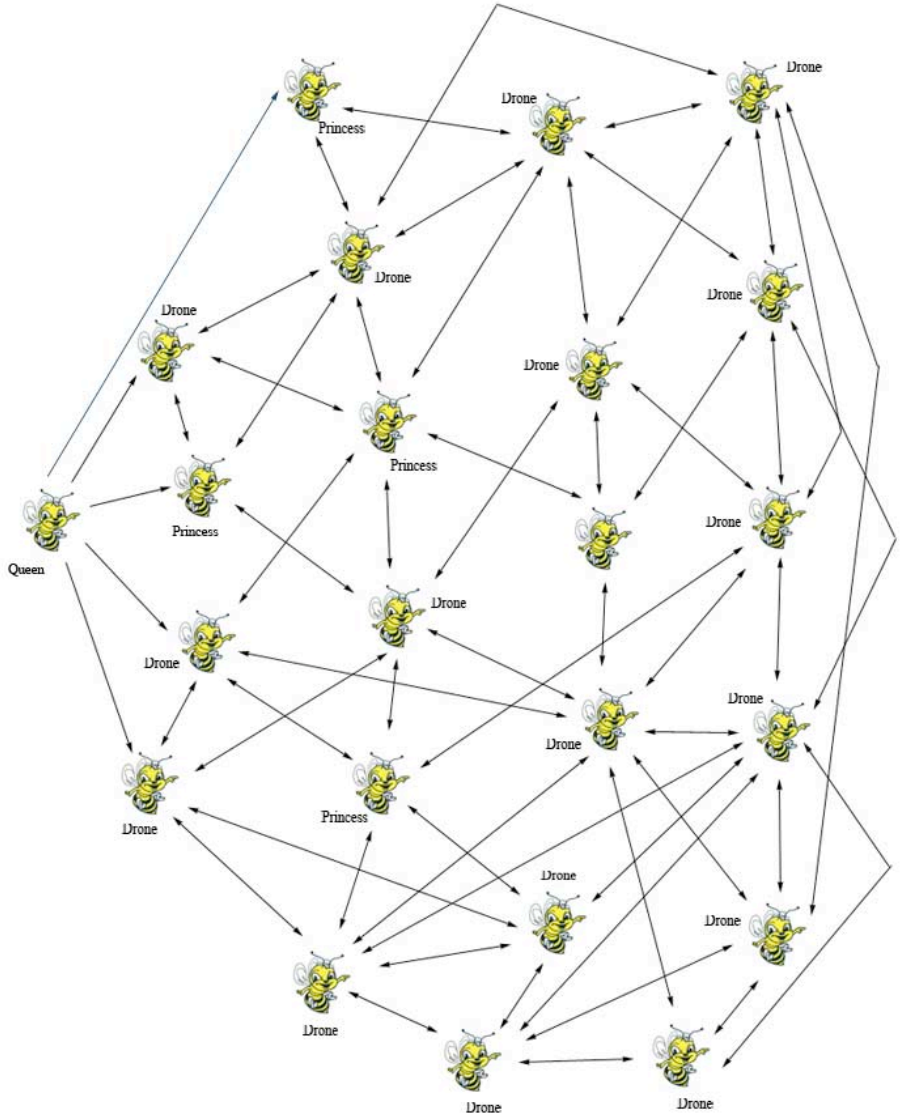


BitTorrent: the difference...



http://www.wtata.com/faq/

BitTorrent at work...



BitTorrent: choosing the pieces

Strict Priority

Use a priority mechanism associated to the chunks

- **Rarest-first:**
  Look at all pieces at all peers, and request piece that's owned by fewest peers
  Increases diversity in the pieces downloaded
  This ensures that the most commonly available pieces are left till the end to download.

- **Random First Piece**:
  When peer starts to download, request random piece.
  (at the beginning is important to get a chunk ASAP)
  When first chunk is obtained, switch to rarest-first

- **End-game mode**:
  Send requests for last sub-chunks to ALL the known peers
  End of download should not be stalled by slow peers
  The idea is to speed up completion of download

BitTorrent: choke/unchoke

Peer serves e.g. 4 (default value) peers in peer set simultaneously

- Seeks best (fastest) downloaders if it's a seeder
- Seeks best uploaders if it's a leecher

Choke is a temporary refusal to upload to a peer

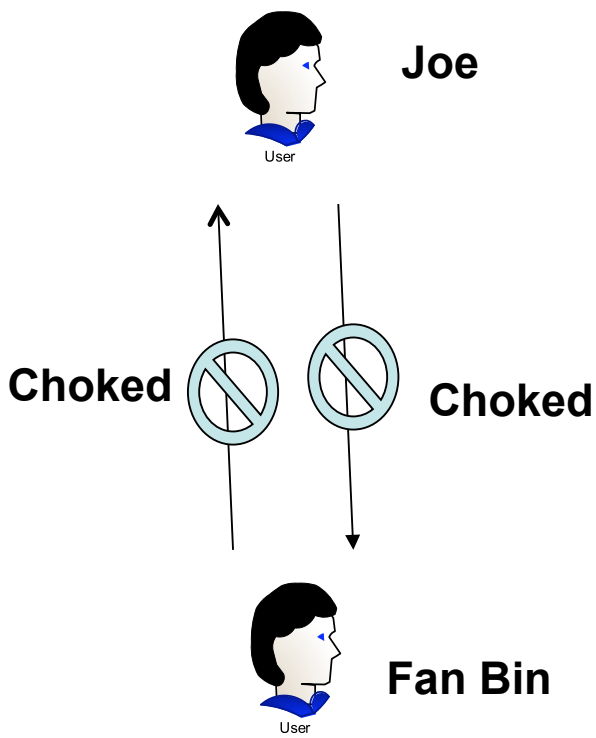
- Leecher serves 4 best uploaders, chokes all others
- Choking evaluation is performed every 10 seconds.
- Each peer unchokes a fixed number of peers (default = 4)
- If there is a better peer, choke the worst of the current 4
- The decision on which peers to un/choke is based solely on download rate, which is evaluated on a rolling, 20-second average

Every 30 seconds peer makes an "**optimistic unchoke**"

- Randomly unchoke a peer from peer set
- Idea:
  - -Maybe it offers better service;
  - -To provide minimal service to new peers

BitTorrent: reasons for choking

- Avoid free-riders
- TCP congestion control.
- To ensure the peers to get a consistent download rate.



Problem: anti-snubbing

- Occasionally peer will be choked by all peers it was previously downloading from.
- If after 1 minute no new pieces obtained then assume snubbed by peer.
- When snubbed, stop uploading to peer.
- Instead do an additional optimistic unchoke.
- Results in faster restoration of download rate.

Upload-only Mode

- Once download is complete, a peer has no download rates to use for comparison nor has any need to use them. The question is, which nodes to upload to?

- Policy:
  - Upload to those with the best upload rate. This ensures that pieces get replicated faster, and new seeders are created fast

BitTorrent: Pros and Cons

Pros:

Proficient in utilizing partially downloaded files
Discourages “freeloading”, by rewarding fastest uploaders
Encourages diversity through “rarest-first”
Works well for “hot content”

Cons:

Assumes all interested peers active at same time;
performance deteriorates if swarm “cools off”
The leech problem.

Dependence on centralized tracker: pro/con?

- Single point of failure: New nodes can’t enter swarm if tracker goes down
- Lack of a search feature:
    - Prevents pollution attacks
    - Users need to resort to out-of-band search: web-search

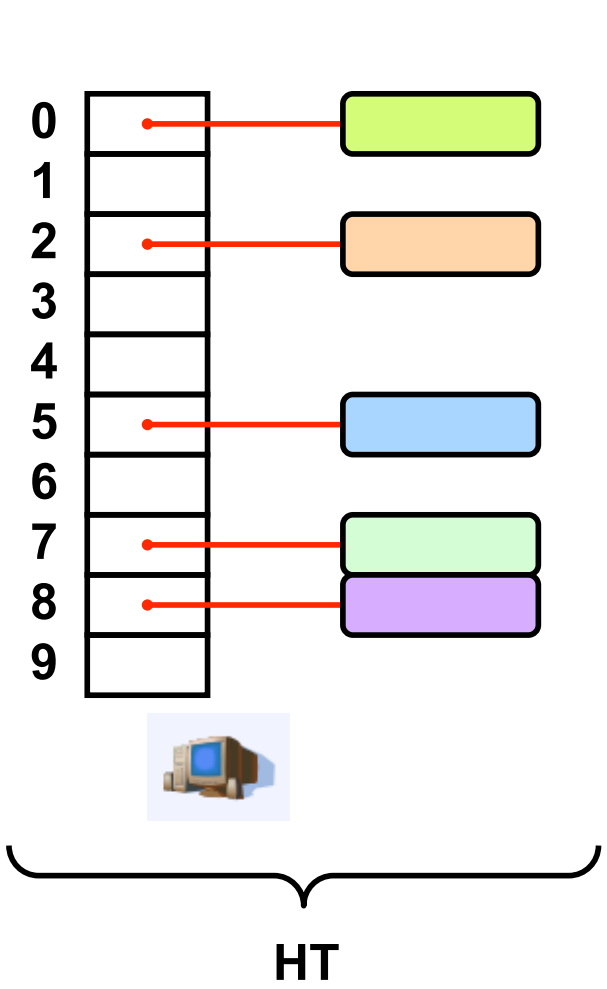
Structured P2P: DHTs

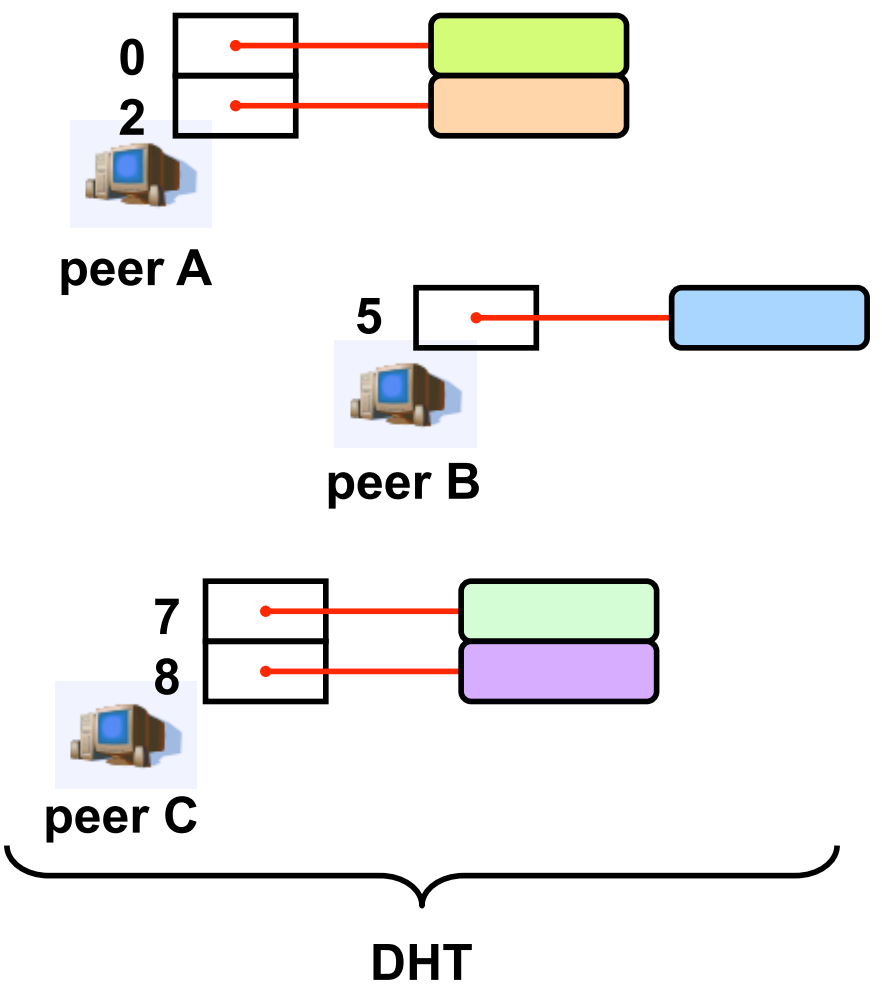
CAN, CHORD, PASTRY; TAPESTRY, ...

Structured vs Unstructured

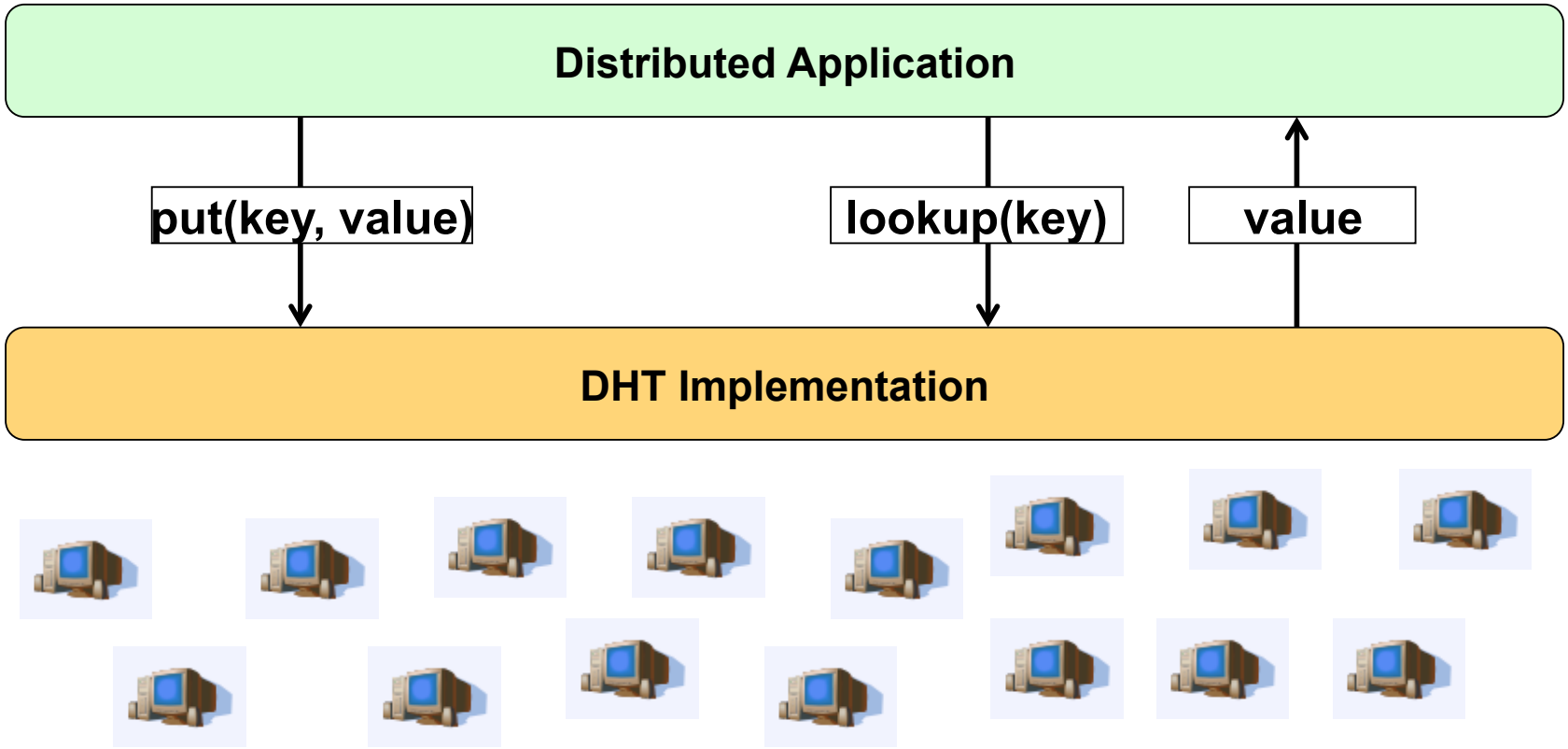
- Unstructured networks/systems
  - Based on searching
  - Unstructured does NOT mean complete lack of structure
    - Network has graph structure
  - But peers are free to join anywhere and objects can be stored anywhere
- Structured networks/systems
  - Based on addressing
  - Network structure determines where peers belong in the network and where objects are stored
  - Should be efficient for locating objects

DHTs





Using the Abstraction



Example of a DHT: Chord

- Let's look at Chord, a famous DHT project
  - Developed at MIT in 2001
  - http://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf

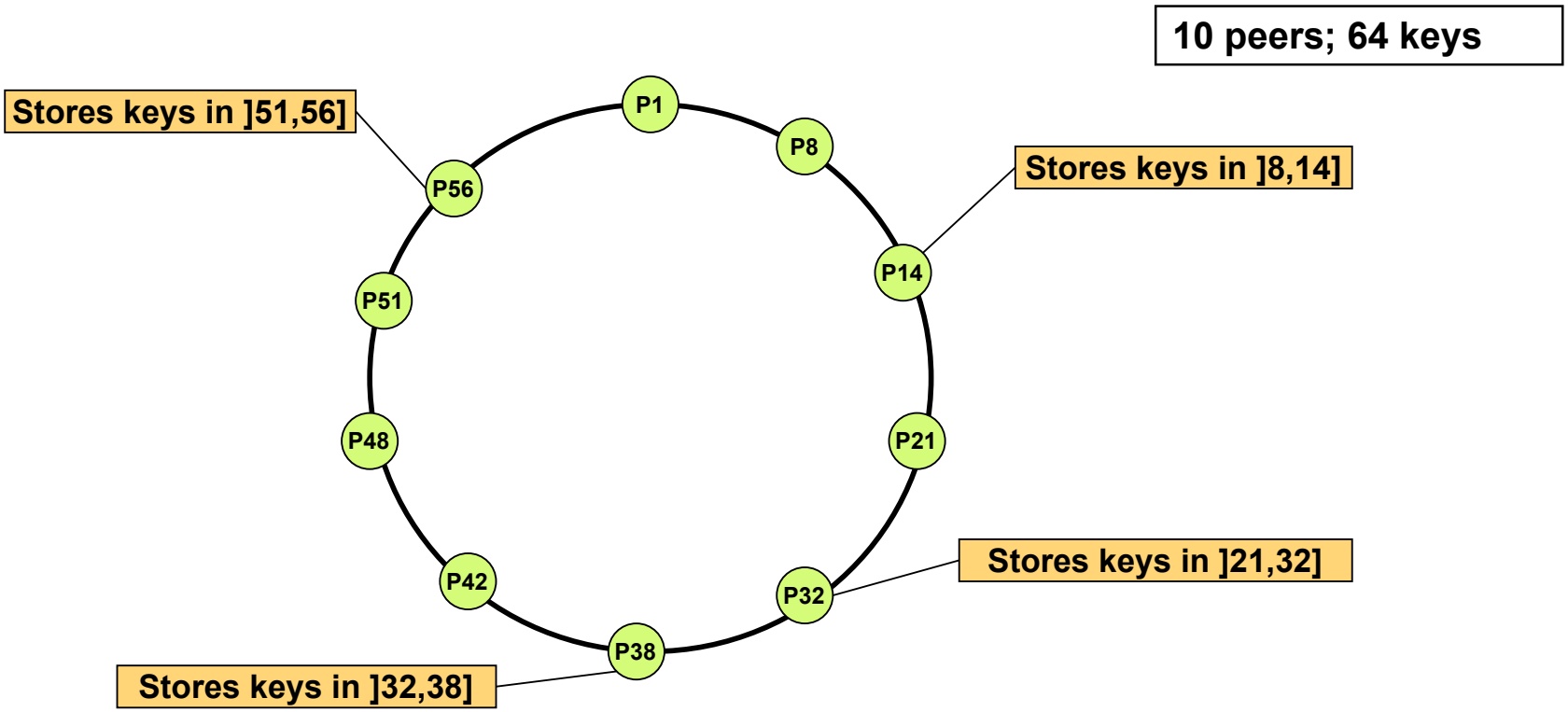
- File names and node names are hashed to the same space, i.e., numbers between 0 and 2^m-1
  - where m is large enough and the hash function good enough that collisions happen only with infinitesimal probability
- Each file has a unique fileID
  - e.g., hash of its name
- Each peer has a unique peerID
  - e.g., hash of its IP address)
- Important: there is no difference between a fileID and a peerID

The Chord Ring

- Peers are organized as a sorted ring
  - Peers are along the ring in increasing order of peerID
    - Remember, peerIDs are just numbers
  - Called a “Chord ring”
- Each peer knows its successor and predecessor in the ring
  - For now let’s assume no churn what-so-ever
  - No peer arrives, no peer departs

- Main Chord idea: A Peer stores Keys that are immediately lower than its peerID
- Let’s look at an example Chord ring and see which peer stores what

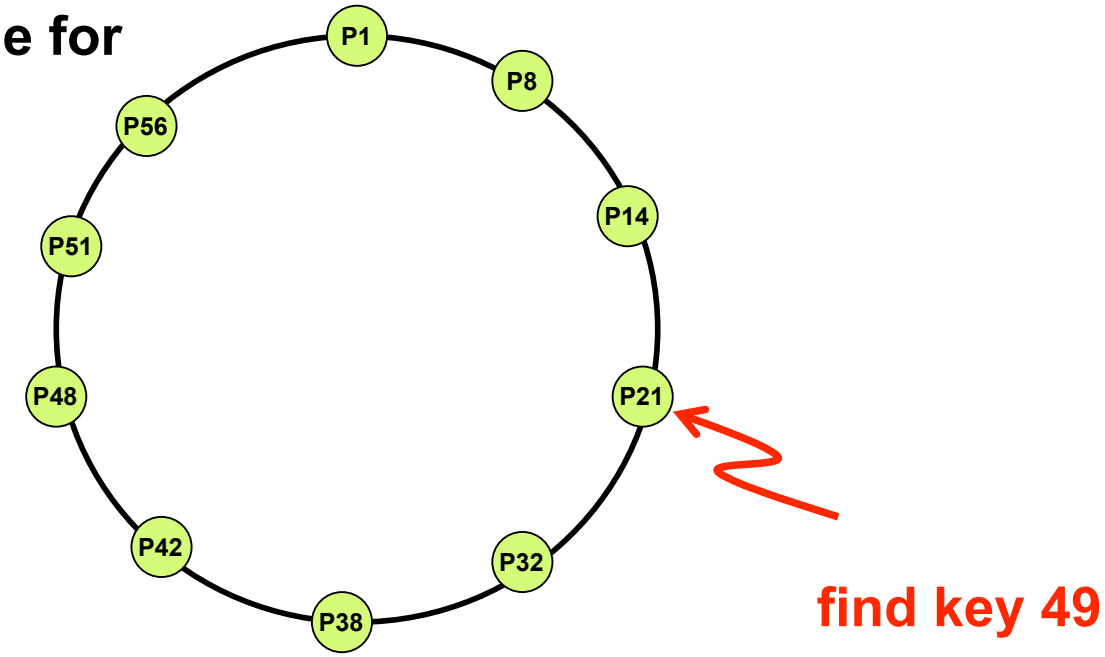
A Chord Ring



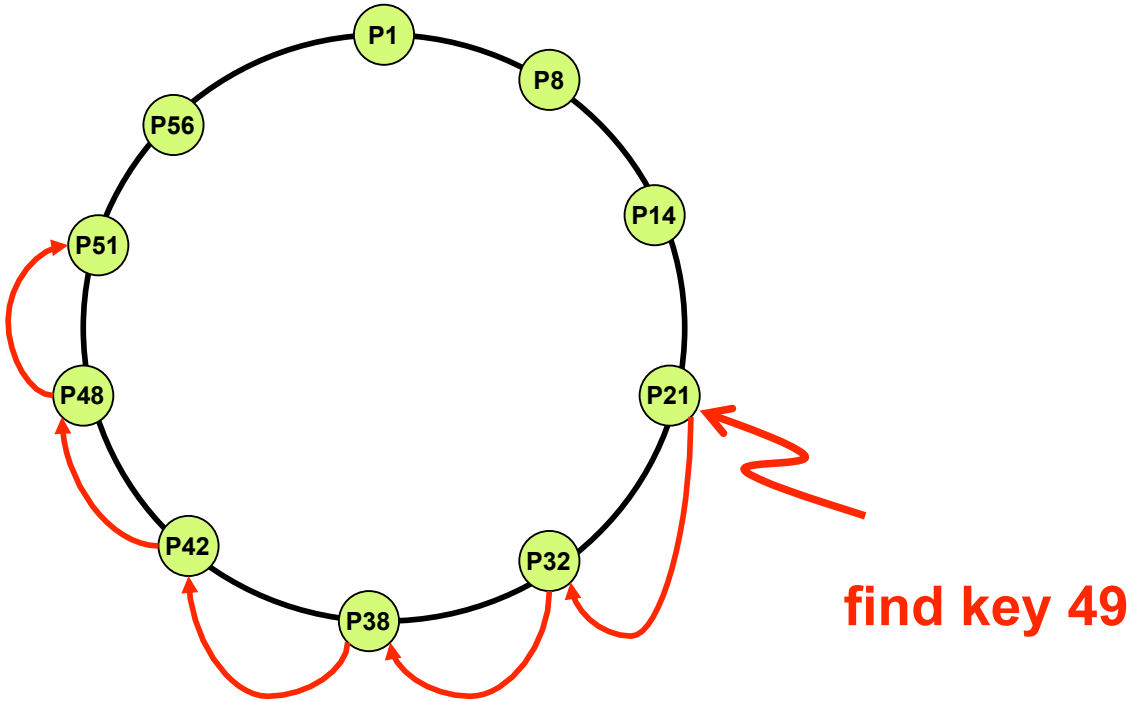
A peer stores (key,value) pairs whose keys are lower than the peer's peerID and higher than the peerID of the peer's predecessor

Put() and Lookup()

Principles is the same for Put() and Lookup()



Put() and Lookup()



Go around the ring, following the “successor” links
Stop at the first peerID that is larger than 49 (peer 51 here)
If key 49 was stored in the network, peer 51 has it

Scalability and Performance

- The Chord ring as we have shown it is very scalable:
  - Each peer only needs to know about two other peers
  - Very small routing table!

- The problem is that the performance is very poor
  - The worst case complexity for a lookup is O(N) hops, where N is the number of peers
  - Since N can be on the order of millions, clearly it's not even remotely acceptable
    - Each hop will take hundreds of milliseconds

- Question: how can we make the number of hops O(log N)?
- Answer: By adding more edges in the network

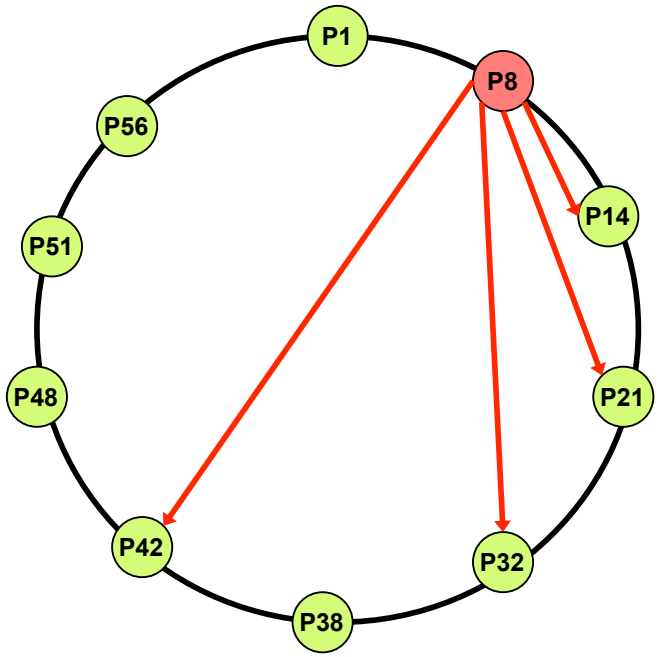
Chord Fingers

- Each peer maintains a “finger table” that contains m entries
  - We have 2^m potential peers in the system
  - So the finger table has at most O(log N) entries

- The i^{th} entry in the finger table of peer A stores the peerID of the first peer B that succeeds A by at least 2^{i-1} on the chord ring
  - B = successor($n+2^{i-1}$)
  - B is called the i^{th} finger of peer A

- Let’s see an example:

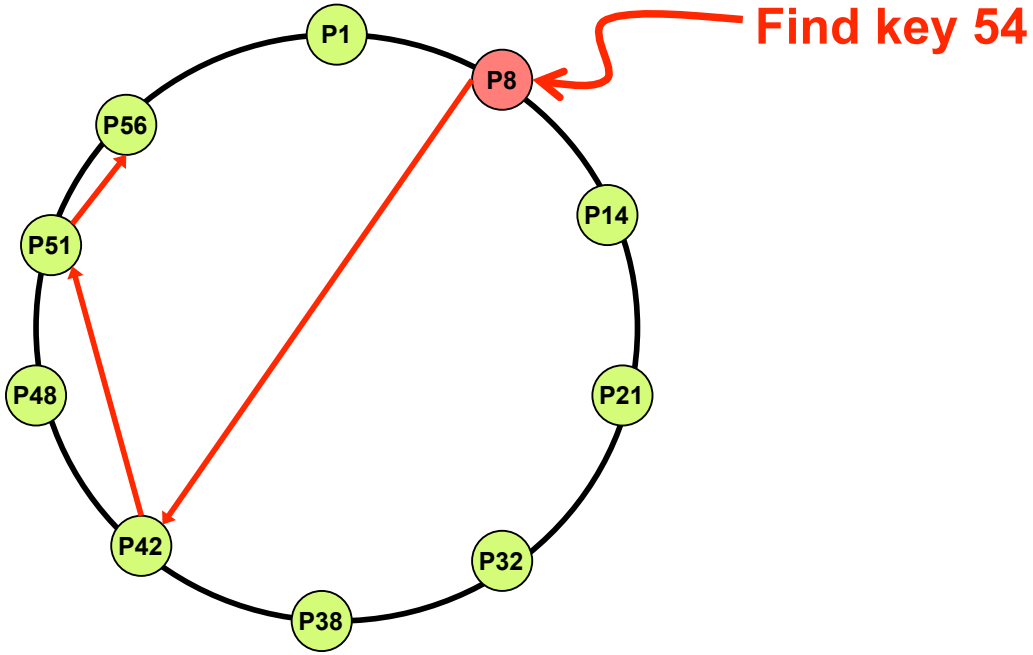
Chord Fingers



Finger Table for P8	
P8+1	P14
P8+2	P14
P8+4	P14
P8+8	P21
P8+16	P32
P8+32	P42

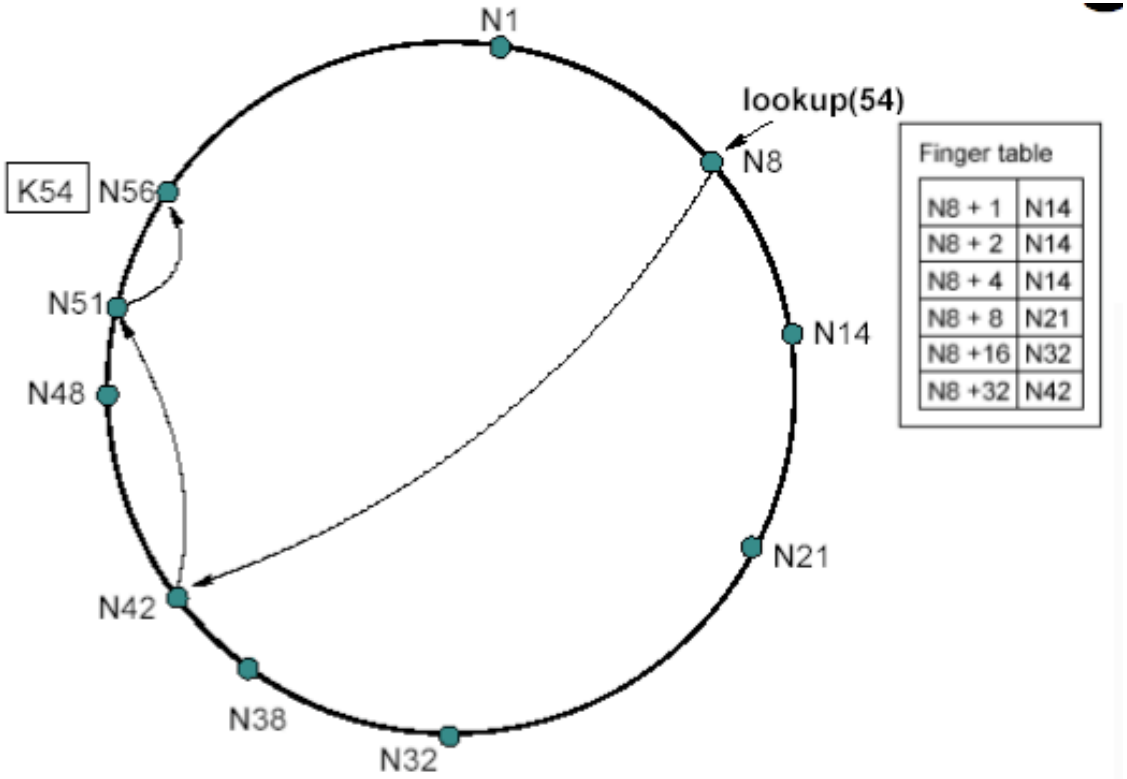
finger[i] = first peer that succeeds peer $(p+2^i)$ mod 2^m

Using Chord Fingers



- With the finger table, a peer can forward a query a least halfway to its destination in one hop
- One can easily prove that the worst case number of hops is O(log N)

Example



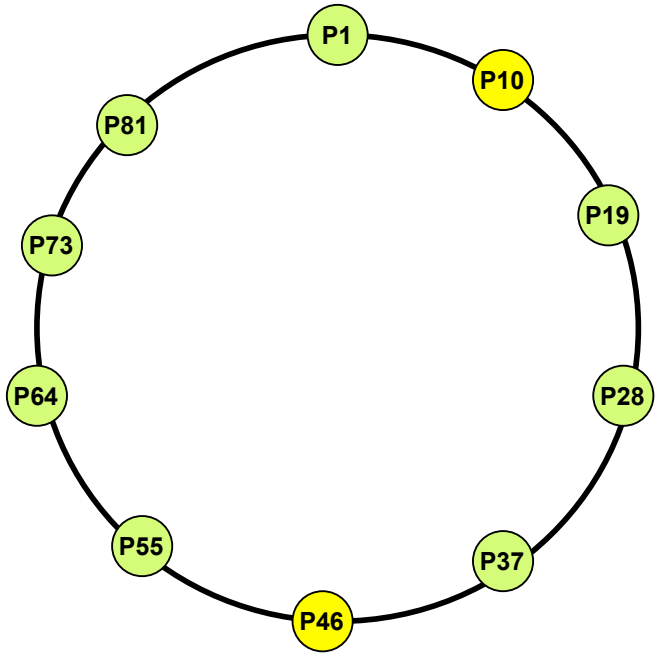
Finger table

N8 + 1	N14
N8 + 2	N14
N8 + 4	N14
N8 + 8	N21
N8 +16	N32
N8 +32	N42

Finger table

N42+ 1	N48
N42+ 2	N48
N42+ 4	N48
N42+ 8	N51
N42+16	N1
N42+32	N14

Chord Fingers



Routing Table for P8	
P10+1	P19
P10+2	P19
P10+4	P19
P10+8	P19
P10+16	P28
P10+32	P46

finger[i] = first peer that succeeds peer $(p+2^i)$ mod 2^m

Bibliography

- Coulouris *et al.*, Distributed Systems: Concepts and Design, 5th Edition, Chapter 10.