Peer-to-Peer (P2P)

CS204: Advanced Computer Networks

Oct 11, 2023

Adapted from Jiasi's CS 204 slides for Spring 23

Overview

- Basics
- Historical P2P
  - Napster
  - Gnutella
  - KaZaA
- Distributed hash tables
  - Basics
  - Chord
  - BitTorrent

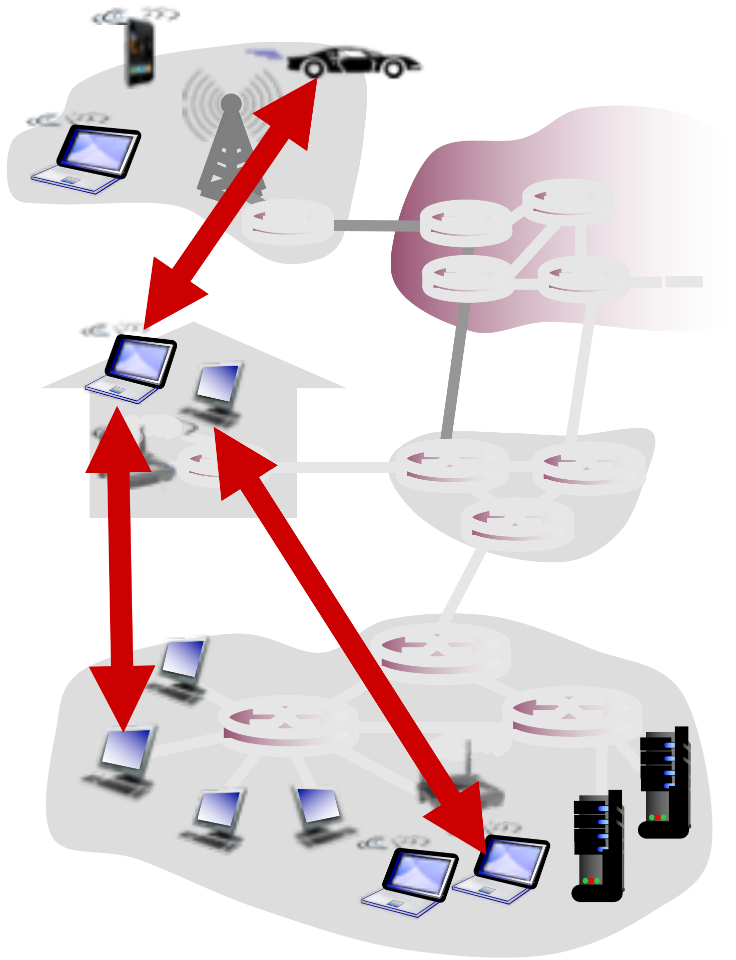
Q: How to share efficiently search for and share files between peers?

Pure P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

examples:

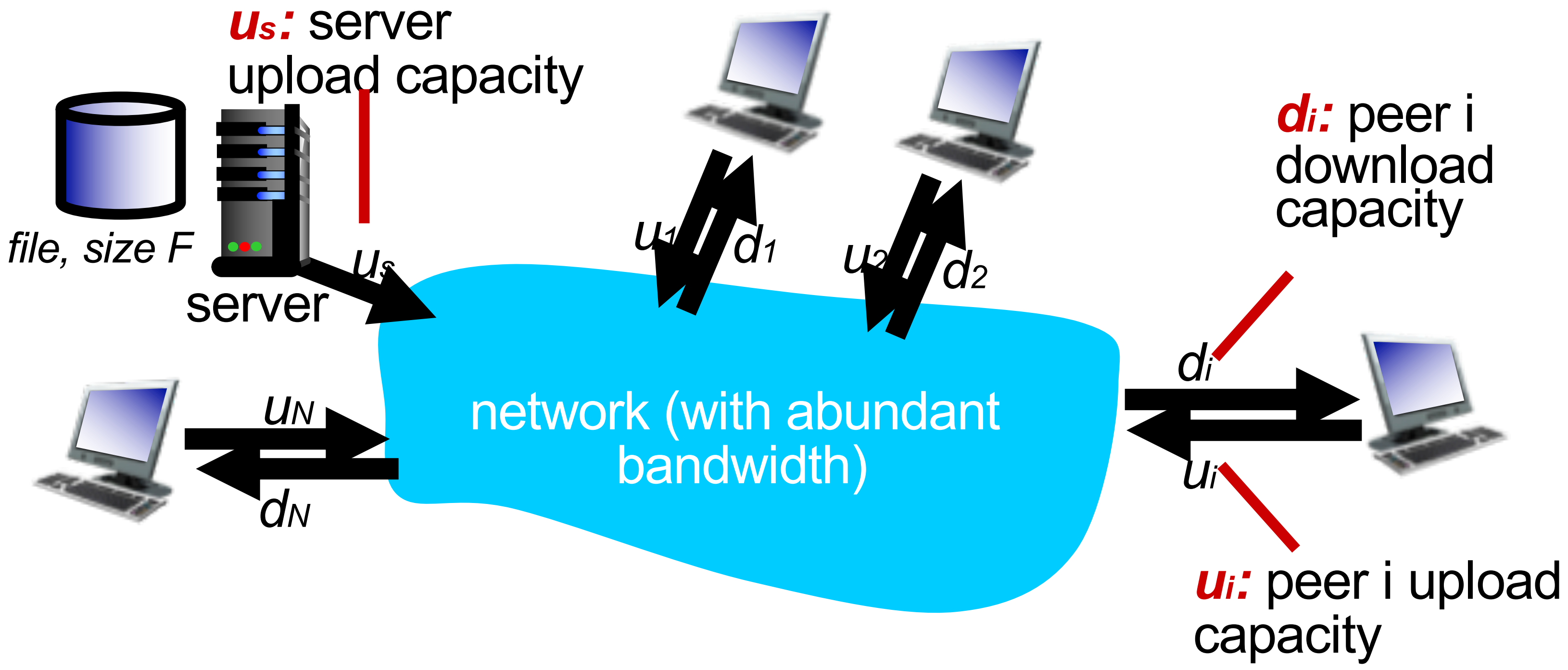
- file distribution (BitTorrent)
- Streaming
- VoIP (original Skype)



File distribution: client-server vs P2P

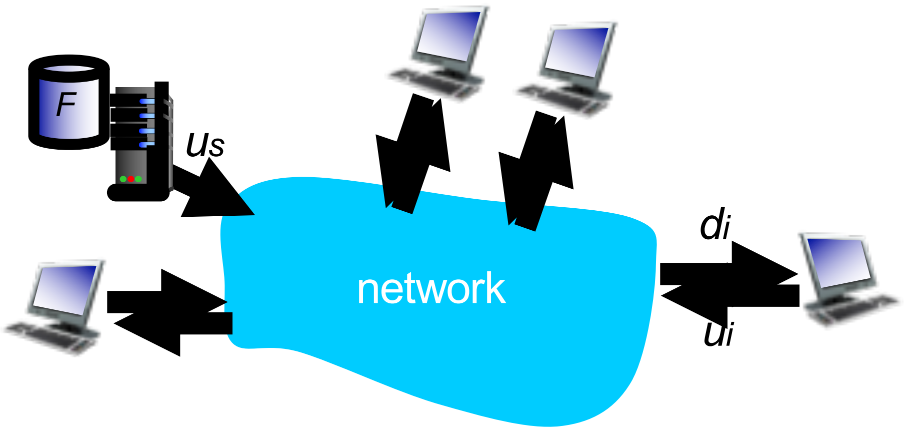
Question: how much time to distribute file (size *F*) from one server to *N* *peers*?

- peer upload/download capacity is limited resource



File distribution time: client-server

- *server transmission:* must sequentially send (upload) *N* file copies:
  - time to send one copy: F/u_s
  - time to send N copies: NF/u_s
- *client:* each client must download file copy
  - d_{min} = min client download rate
  - min client download time: F/d_{min}



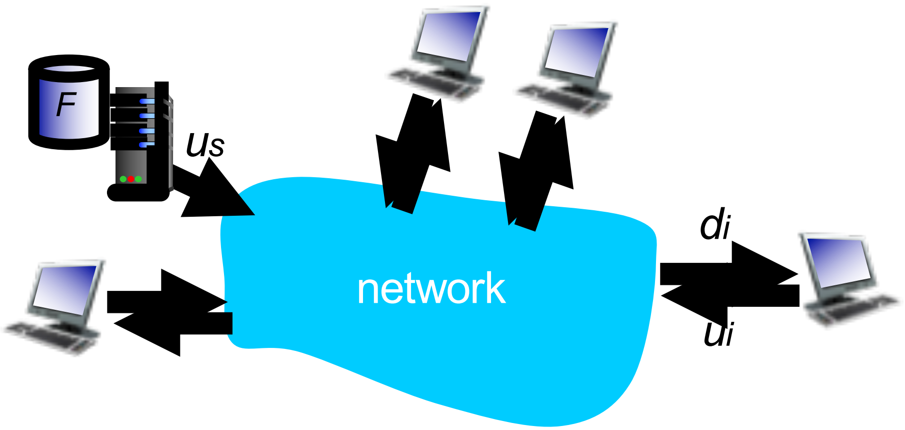
time to distribute F to N clients using client-server approach

$$D_{c\text{-}s} \geq max\{NF/u_s, F/d_{min}\}$$

increases linearly in N

File distribution time: P2P

- *server transmission:* must upload at least one copy
  - time to send one copy: F/u_s
- *client:* each client must download file copy
  - min client download time: F/d_{min}



- *clients:* as aggregate must download *NF* bits
  - max upload rate (limiting max download rate) is $u_s + \Sigma u_i$

time to distribute F to N clients using P2P approach

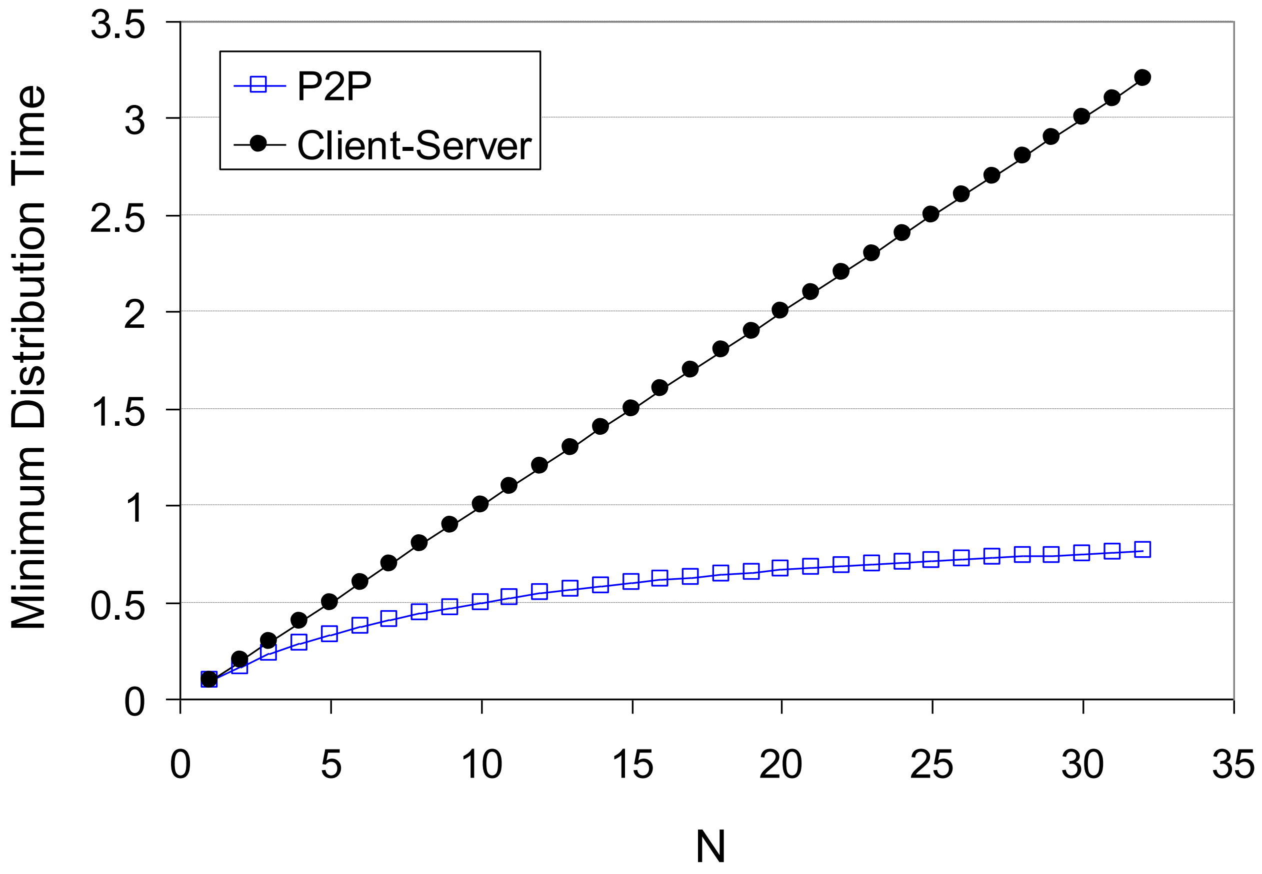
$$D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

increases linearly in *N* …

… but so does this, as each peer brings service capacity

Client-server vs. P2P: example

client upload rate = u, F/u = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$

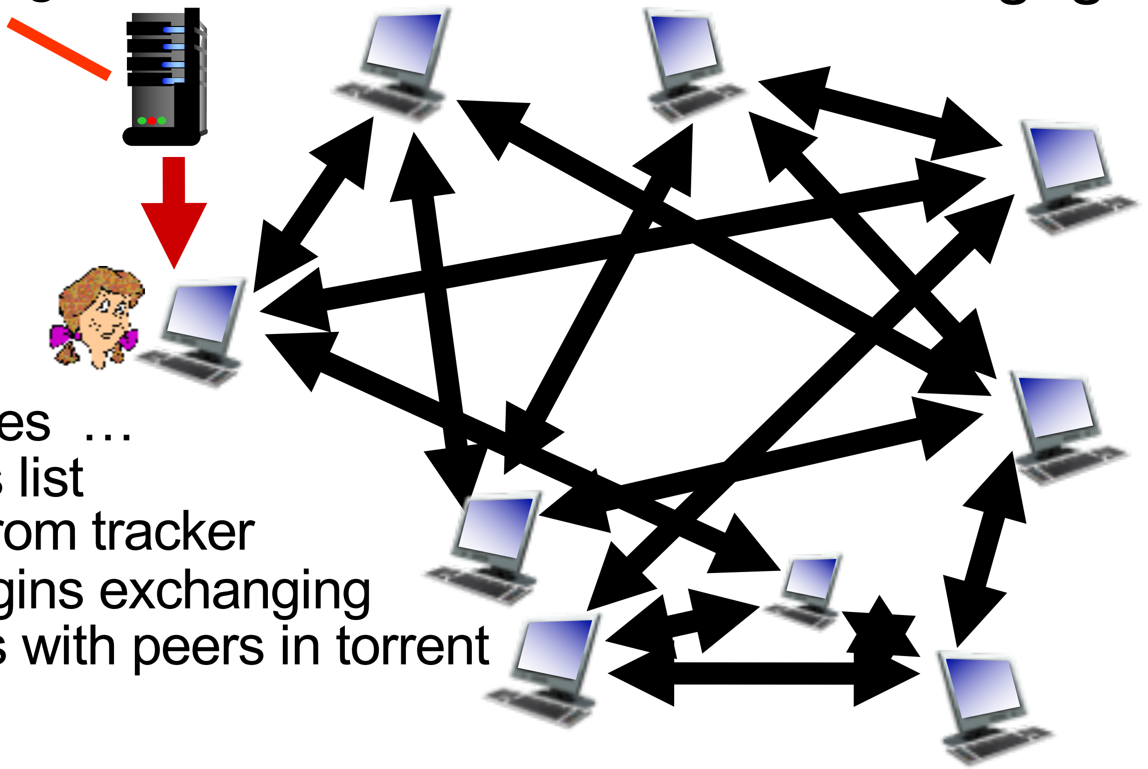


P2P file distribution

- file divided into 256Kb chunks (for example)
- peers in torrent send/receive file chunks

tracker: tracks peers participating in torrent

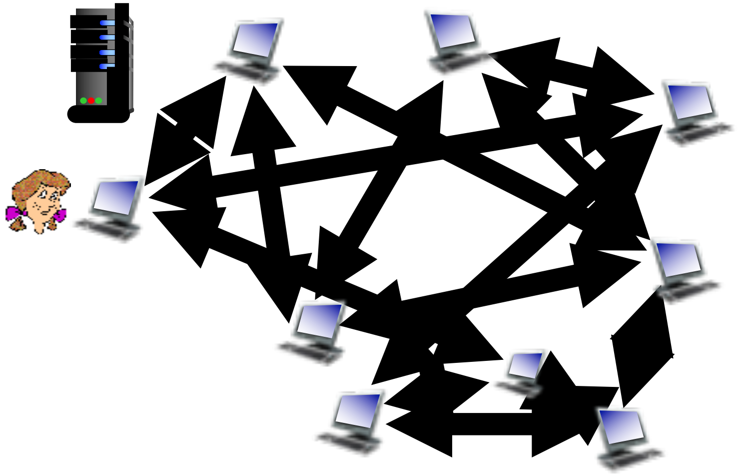
torrent: group of peers exchanging chunks of a file



Alice arrives …
… obtains list
of peers from tracker
… and begins exchanging
file chunks with peers in torrent

P2P file distribution

- peer joining torrent:
  - has no chunks, but will accumulate them over time from other peers
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")



- while downloading, peer uploads chunks to other peers
- peer may change peers with whom it exchanges chunks
- *churn:* peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

What can P2P teach us about infrastructure design?

- Resistant to DoS and failures
  - Safety in numbers, no single point of failure
- Self-assembling
  - Nodes insert themselves into structure
  - No manual configuration or oversight
- Flexible: nodes can be
  - Widely distributed or colocated
  - Powerful hosts or low-end PCs
- Each peer brings a little bit to the dance
  - Aggregate is equivalent to a big distributed server farm behind a fat network pipe

General Abstraction?

- Big challenge for P2P: finding content
  - Many machines, must find one that holds data
- Essential task: lookup(key)
  - Given key, find host that has data ("value") corresponding to that key

Overview

- Basics
- Historical P2P
    - Napster
    - Gnutella
    - KaZaA
- Distributed hash tables
    - Basics
    - Chord
    - BitTorrent

Q: How to share efficiently search for and share files between peers?

Locating the Relevant Peers

- Three main approaches
  - Central directory (e.g., Napster)
  - Query flooding (e.g., Gnutella)
  - Hierarchical overlay (e.g., Kazaa, modern Gnutella)
  - Distributed hash table (e.g., BitTorrent)

- Design goals
  - Scalability
  - Simplicity
  - Robustness
  - Plausible deniability

Peer-to-Peer Networks: Napster

- Napster history: the rise
  - 1/99: Napster version 1.0
  - 5/99: company founded
  - 12/99: first lawsuits
  - 2000: 80 million users

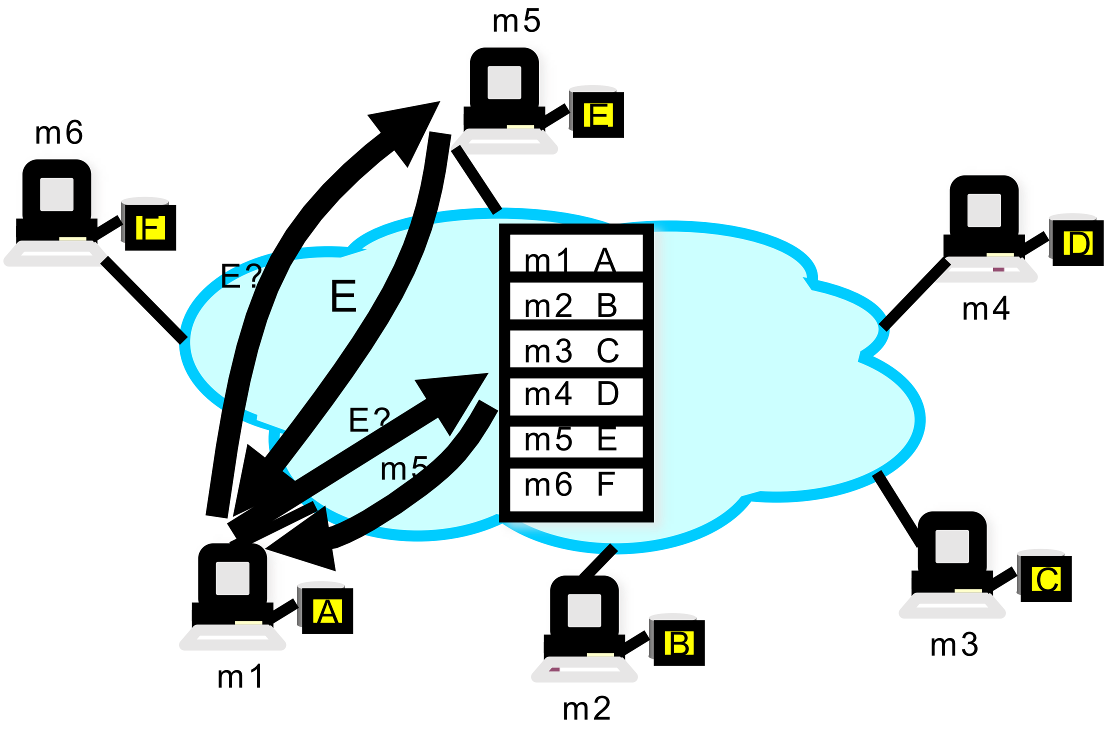
Shawn Fanning, Northeastern freshman

- Napster history: the fall
  - Mid 2001: out of business due to lawsuits
  - Mid 2001: dozens of decentralized P2P alternatives
  - 2003: growth of pay services like iTunes

Napster Directory Service

- Client contacts Napster (via TCP)
  - Provides a list of music files it will share
  - … and Napster's central server updates the directory
- Client searches on a title or performer
  - Napster identifies online clients with the file
  - … and provides their IP addresses
- Client requests the file from the chosen supplier
  - Supplier transmits the file to the client
  - Both client and supplier report status to Napster

Napster: Example



Napster Properties

- Server's directory continually updated
  - Always know what music is currently available
  - Point of vulnerability for legal action
- Peer-to-peer file transfer
  - No load on the server
  - Plausible deniability for legal action (but not enough)
- Bandwidth
  - Suppliers ranked by apparent bandwidth and response time

Napster: Limitations of Directory

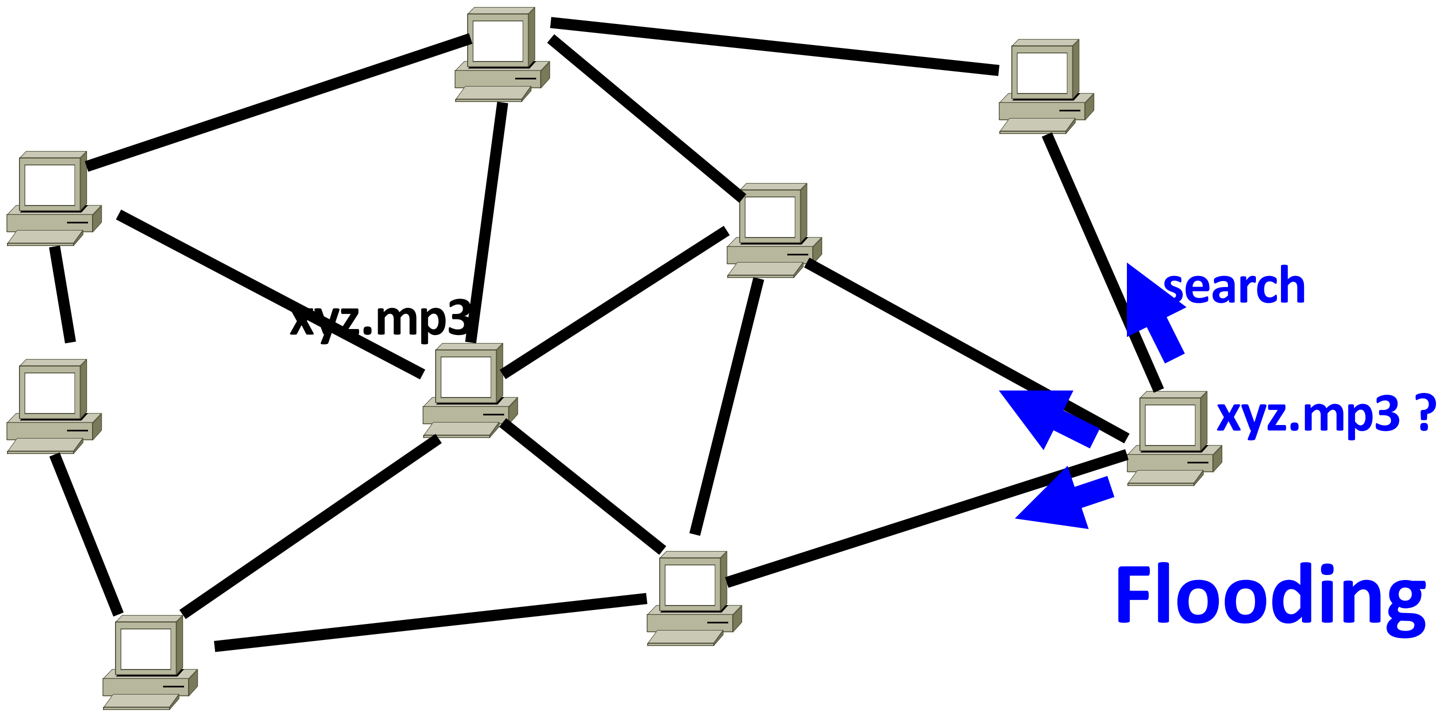
- File transfer is decentralized, but locating content is highly centralized
  - Single point of failure
  - Performance bottleneck
  - Copyright infringement

- So, later P2P systems were more distributed
  - Gnutella went to the other extreme…

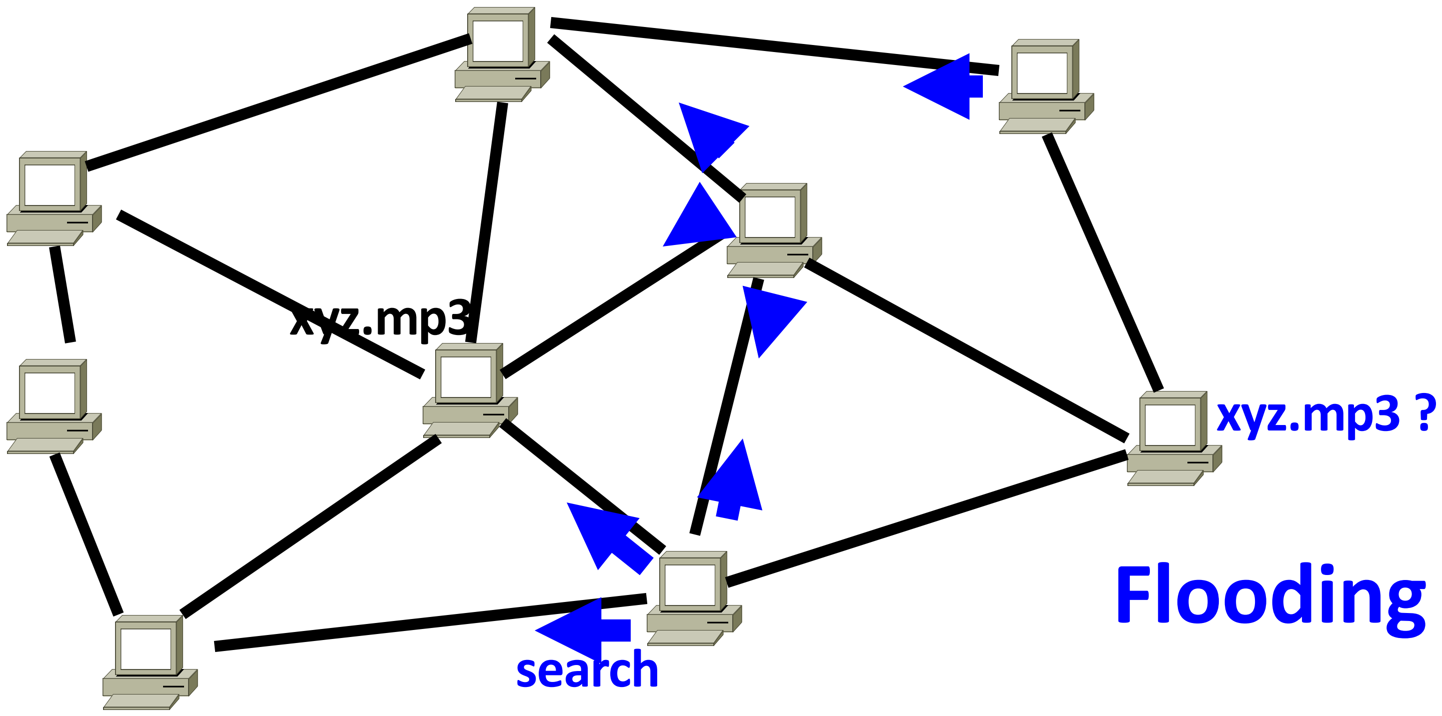
Peer-to-Peer Networks: Gnutella

- Gnutella history
  - 2000: J. Frankel & T. Pepper released Gnutella
  - Soon after: many other clients (e.g., Morpheus, Limewire, Bearshare)
  - 2001: protocol enhancements, e.g., “ultrapeers”
- Query flooding
  - Join: contact a few nodes to become neighbors
  - Publish: no need!
  - Search: ask neighbors, who ask their neighbors
  - Fetch: get file directly from another node

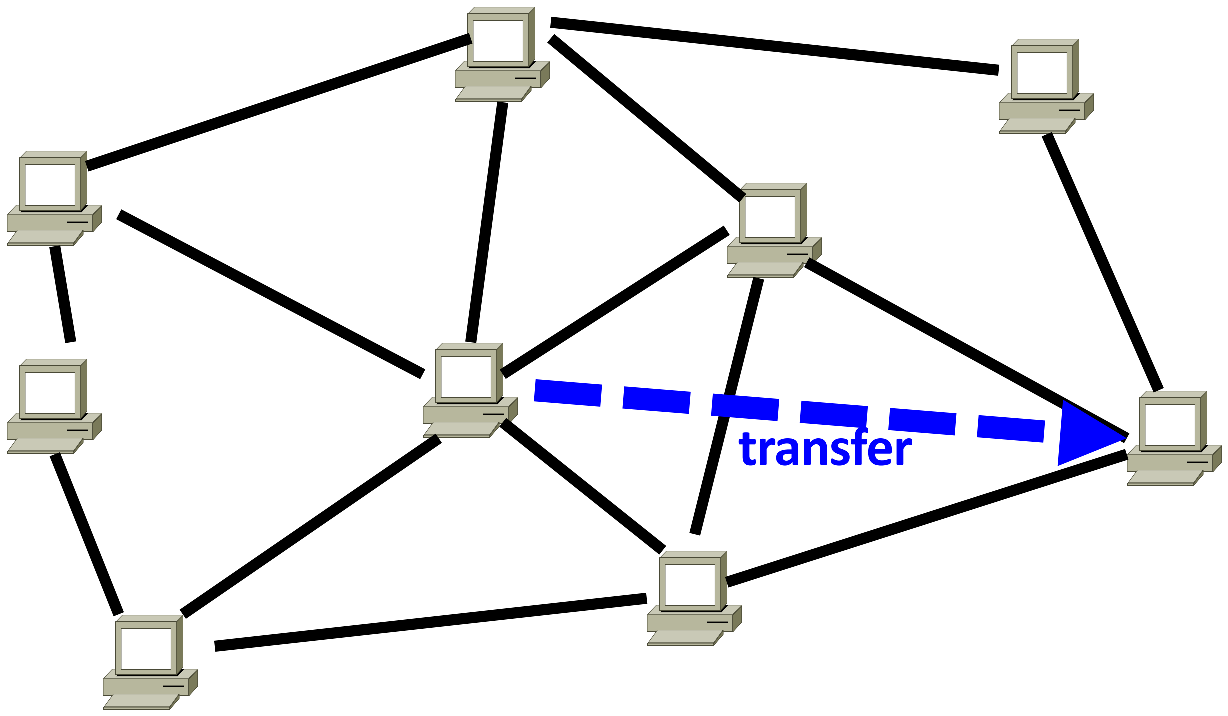
Gnutella: Search by Flooding



Gnutella: Search by Flooding



Gnutella: Search by Flooding



Gnutella: Pros and Cons

- Advantages
  - Fully decentralized
  - Search cost distributed
  - Processing per node permits powerful search semantics

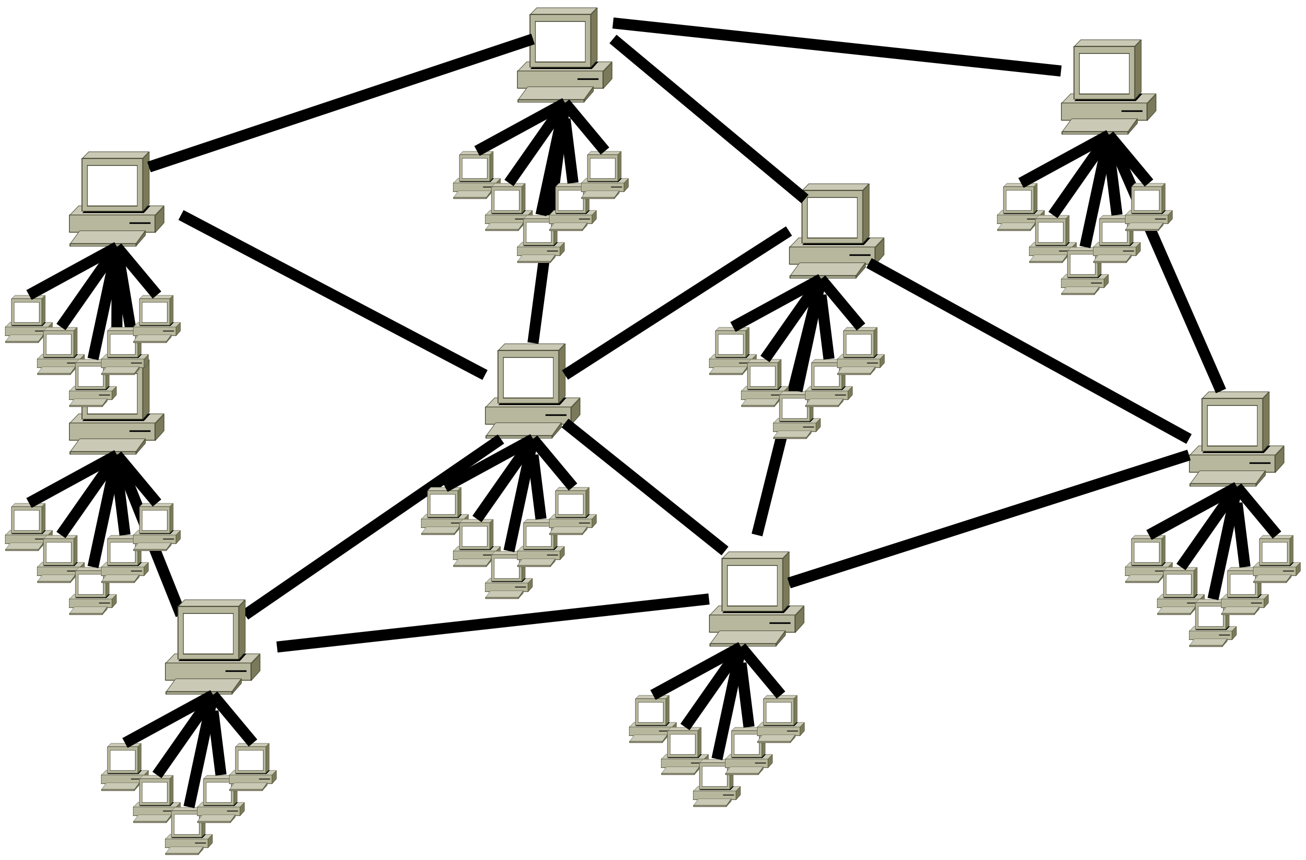
- Disadvantages
  - Search scope may be quite large
  - Search time may be quite long
  - High overhead, and nodes come and go often

Peer-to-Peer Networks: KaZaA

- KaZaA history
  - 2001: created by Dutch company (Kazaa BV)
  - Single network called FastTrack used by other clients as well
  - Eventually protocol changed so others could no longer use it

- Super-node hierarchy
  “not all peers are created equal”
  - Join: on start, the client contacts a super-node
  - Publish: client sends list of files to its super-node
  - Search: queries flooded among super-nodes
  - Fetch: get file directly from one or more peers

"Ultra/super peers" in KaZaA and later Gnutella



KaZaA: Why Super-Nodes?

- Query consolidation
  - Many connected nodes may have only a few files
  - Propagating query to a sub-node may take more time than for the super-node to answer itself
- Stability
  - Super-node selection favors nodes with high up-time
  - How long you've been on is a good predictor of how long you'll be around in the future

Overview

- Basics
- Historical P2P
  - Napster
  - Gnutella
  - KaZaA
- Distributed hash tables
  - Basics
  - Chord
  - BitTorrent

Q: How to share efficiently search for and share files between peers?

Peer-to-Peer Networks: BitTorrent

- BitTorrent history
  - 2002: B. Cohen debuted BitTorrent

- Emphasis on efficient fetching, not searching
  - Distribute same file to many peers
  - Single publisher, many downloaders

- Preventing free-loading
  - Incentives for peers to contribute

BitTorrent™

BitTorrent: Tracker

- Infrastructure node
  - Keeps track of peers participating in the torrent
  - Peers register with the tracker when it arrives
- Tracker selects peers for downloading
  - Returns a random set of peer IP addresses
  - So the new peer knows who to contact for data
- Can also have “trackerless” system
  - Using distributed hash tables (DHTs)

Simple Database

Simple database with (key, value) pairs:

- key: human name; value: social security #

Key	Value
John Washington	132-54-3570
Diana Louise Jones	761-55-3791
Xiaoming Liu	385-41-0902
Rakesh Gopal	441-89-1956
Linda Cohen	217-66-5609
…….	………
Lisa Kobayashi	177-23-0199

- key: movie title; value: IP addresses of clients who have the content

Hash Table

- More convenient to store and search on numerical representation of key
- key = hash(original key)

Original Key	Key	Value
John Washington	8962458	132-54-3570
Diana Louise Jones	7800356	761-55-3791
Xiaoming Liu	1567109	385-41-0902
Rakesh Gopal	2360012	441-89-1956
Linda Cohen	5430938	217-66-5609
…….		………
Lisa Kobayashi	9290124	177-23-0199

Distributed Hash Table (DHT)

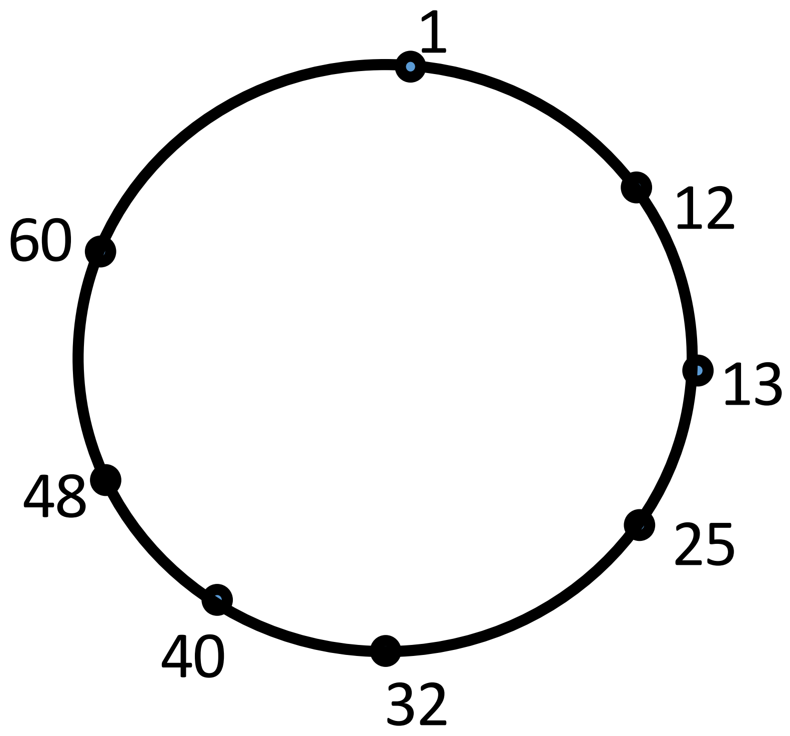
- Distribute (key, value) pairs over millions of peers
  - pairs are evenly distributed over peers
- Any peer can query database with a key
  - database returns value for the key
  - To resolve query, small number of messages exchanged among peers
- Each peer only knows about a small number of other peers
- Robust to peers coming and going (churn)

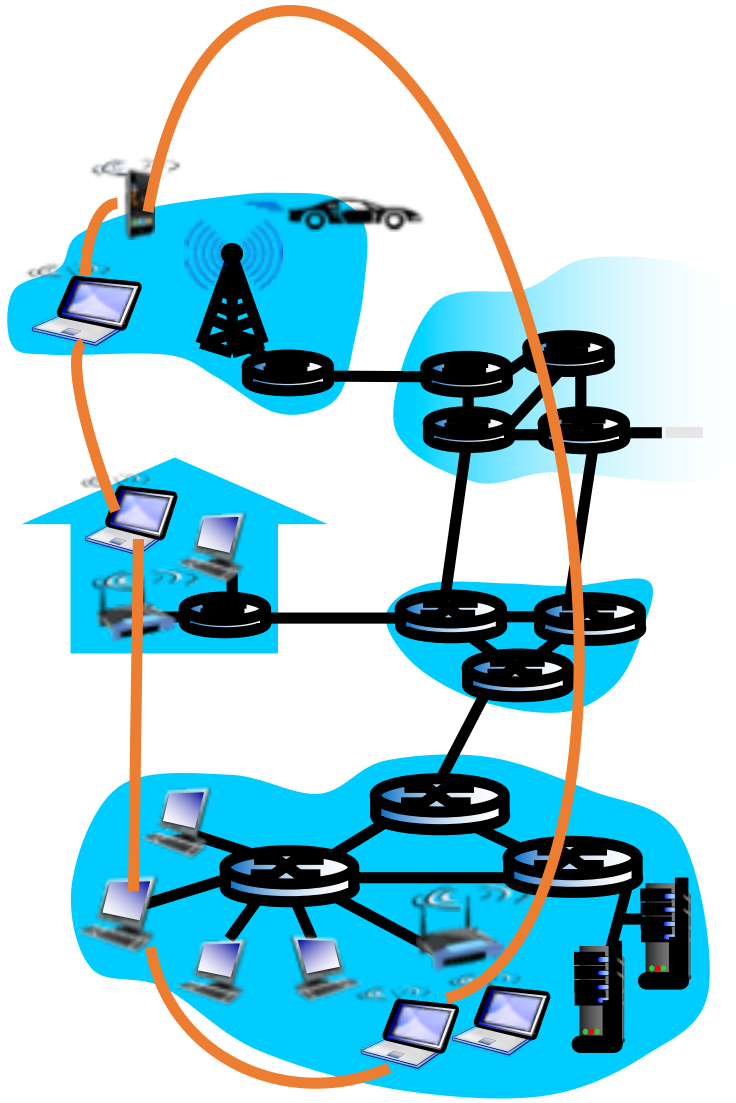
Assign key-value pairs to peers

- rule: assign key-value pair to the peer that has the closest ID
- convention: closest is the immediate successor of the key
- e.g., ID space {0,1,2,3,…,63}
- suppose 8 peers: 1,12,13,25,32,40,48,60
  - If key = 51, then assigned to peer 60
  - If key = 60, then assigned to peer 60
  - If key = 61, then assigned to peer 1

Circular DHT

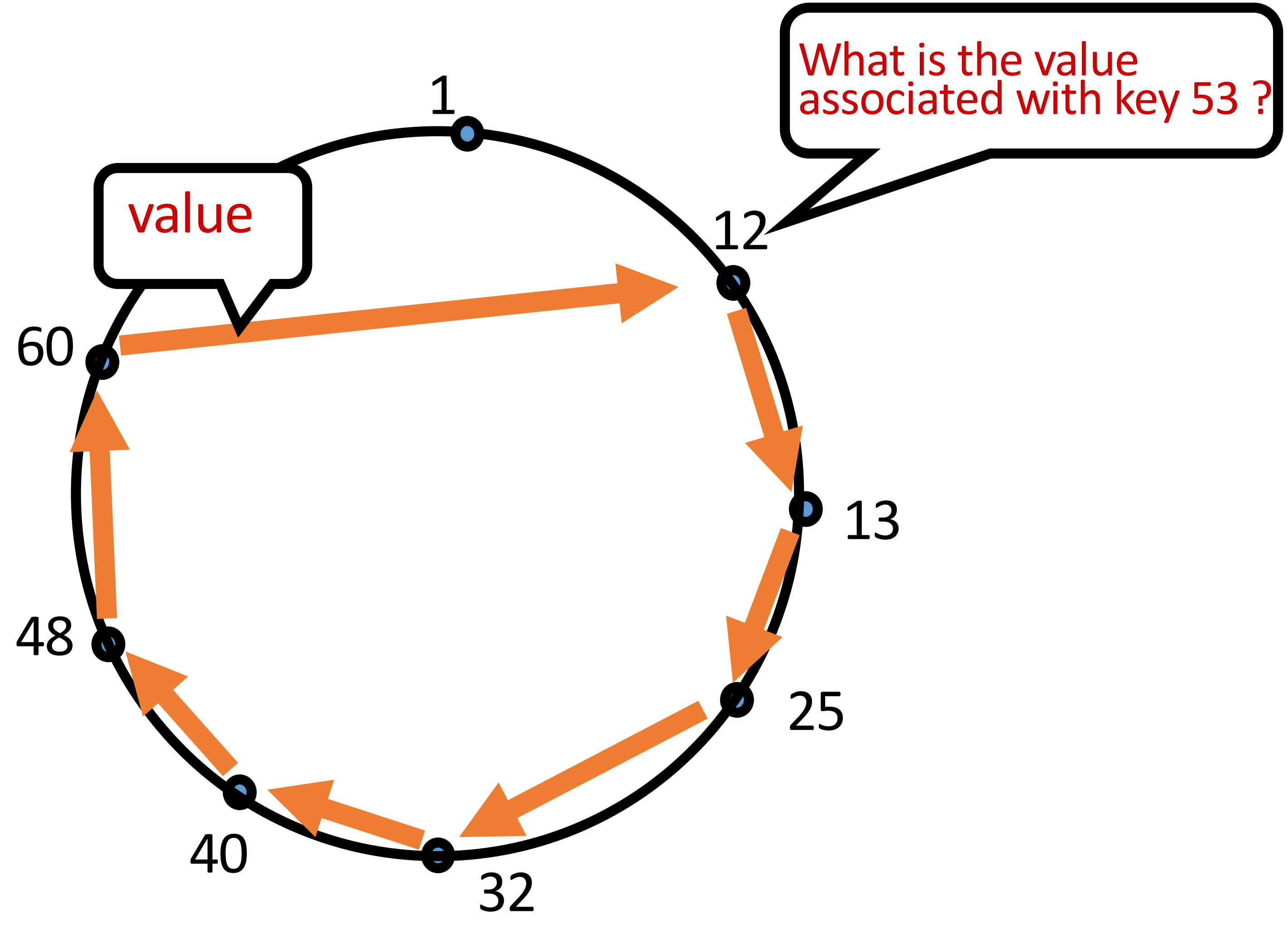
- each peer *only* aware of immediate successor and predecessor.



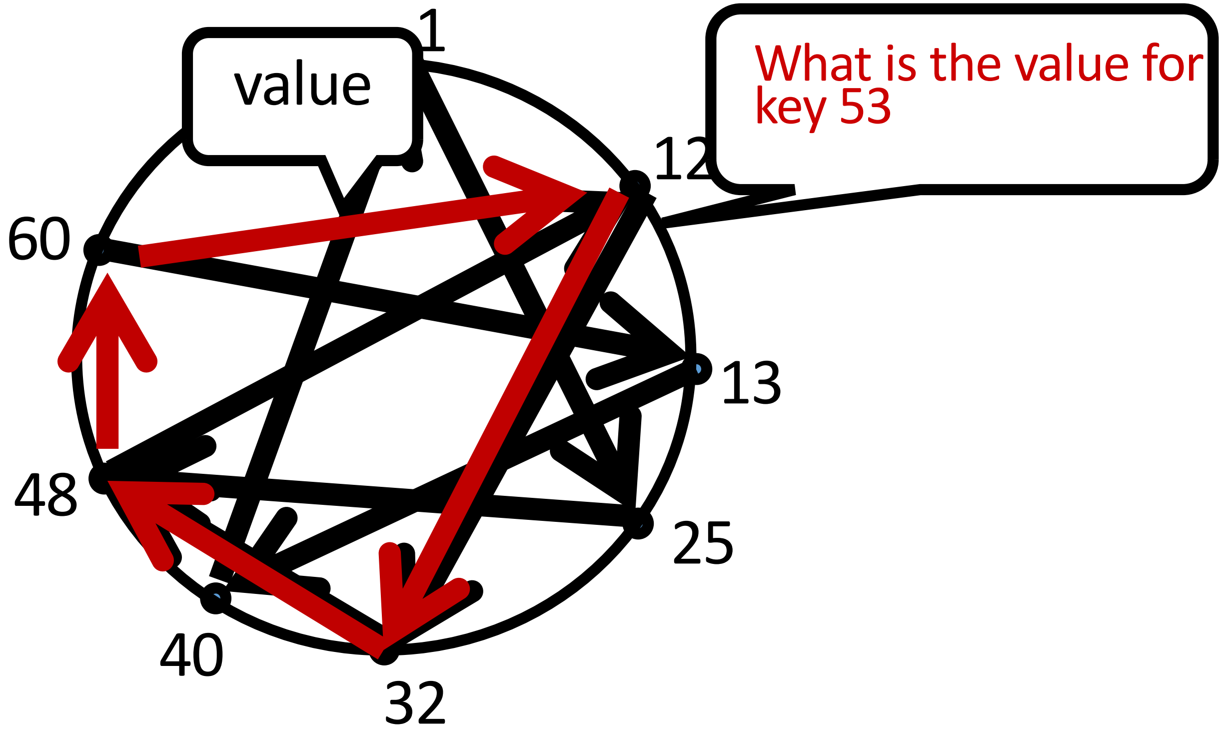


Overlay onto real network

Resolving a query



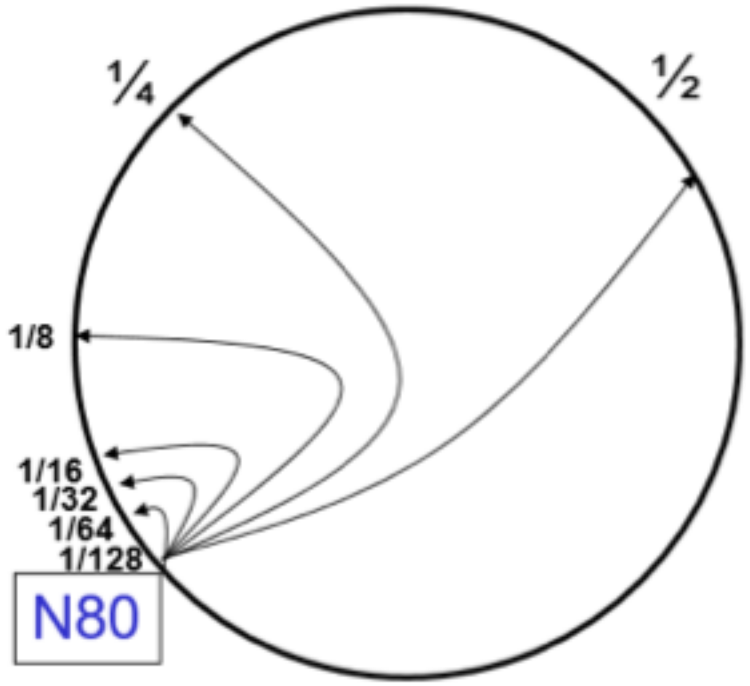
Circular DHT with shortcuts



- each peer keeps track of IP addresses of predecessor, successor, short cuts.
- reduced from 6 to 3 messages.
- possible to design shortcuts with *O(log N)* neighbors, *O(log N)* messages in query

Chord: Fast routing with a small routing table

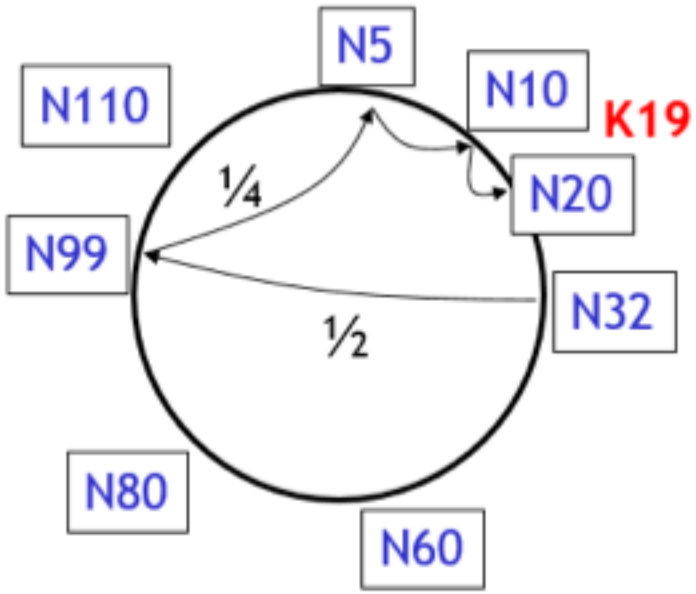
- Each node's routing table lists nodes:
  - ½ way around circle
  - ¼ way around circle
  - …
  - next around circle
- The table is small:
  - At most log N entries



Chord: Lookups take O(log N) hops

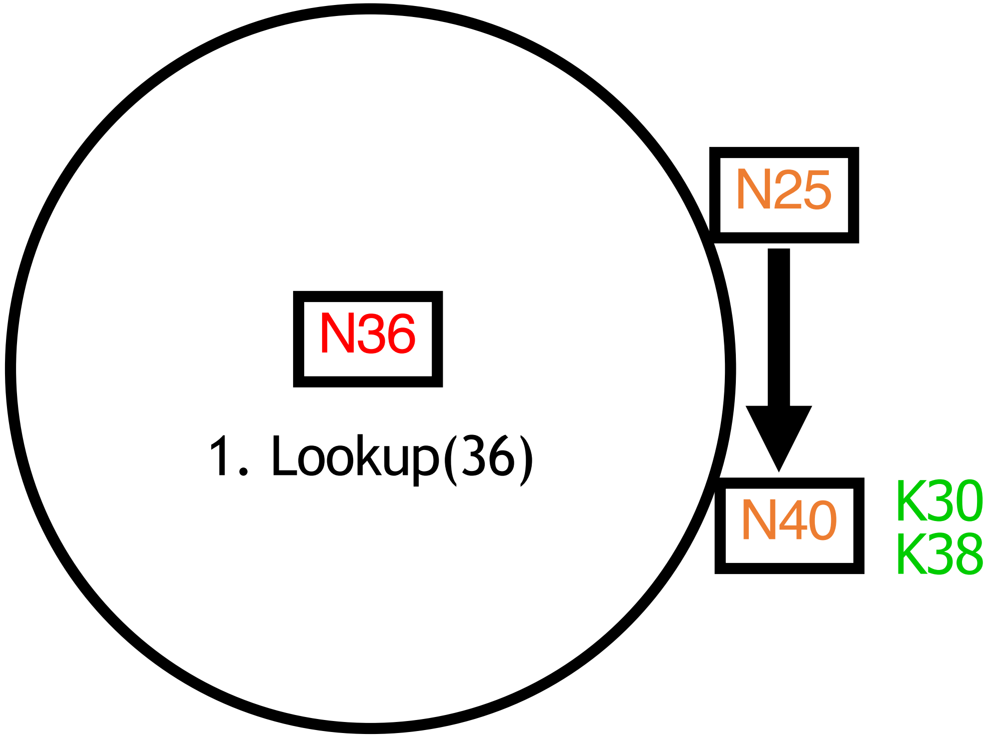
- Every step reduces the remaining distance to the destination by at least a factor of 2

- Lookups are fast:
  - At most O(log *N*) steps
  - Can be made even faster in practice

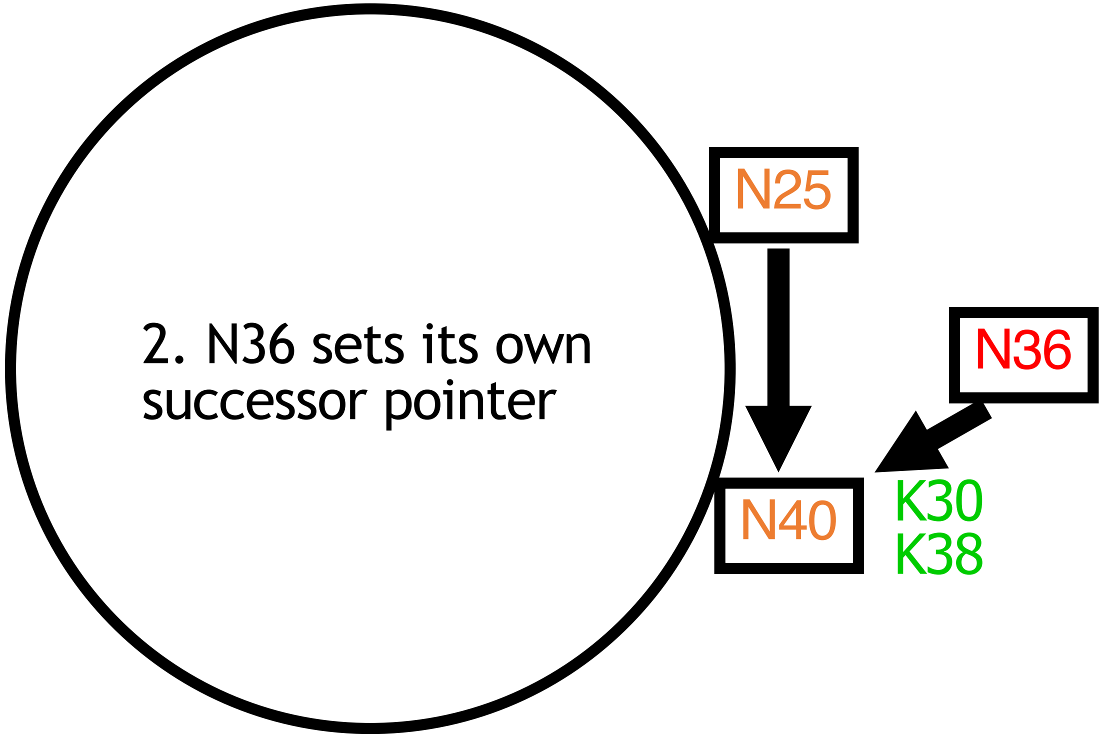


Node N32 looks up key K19

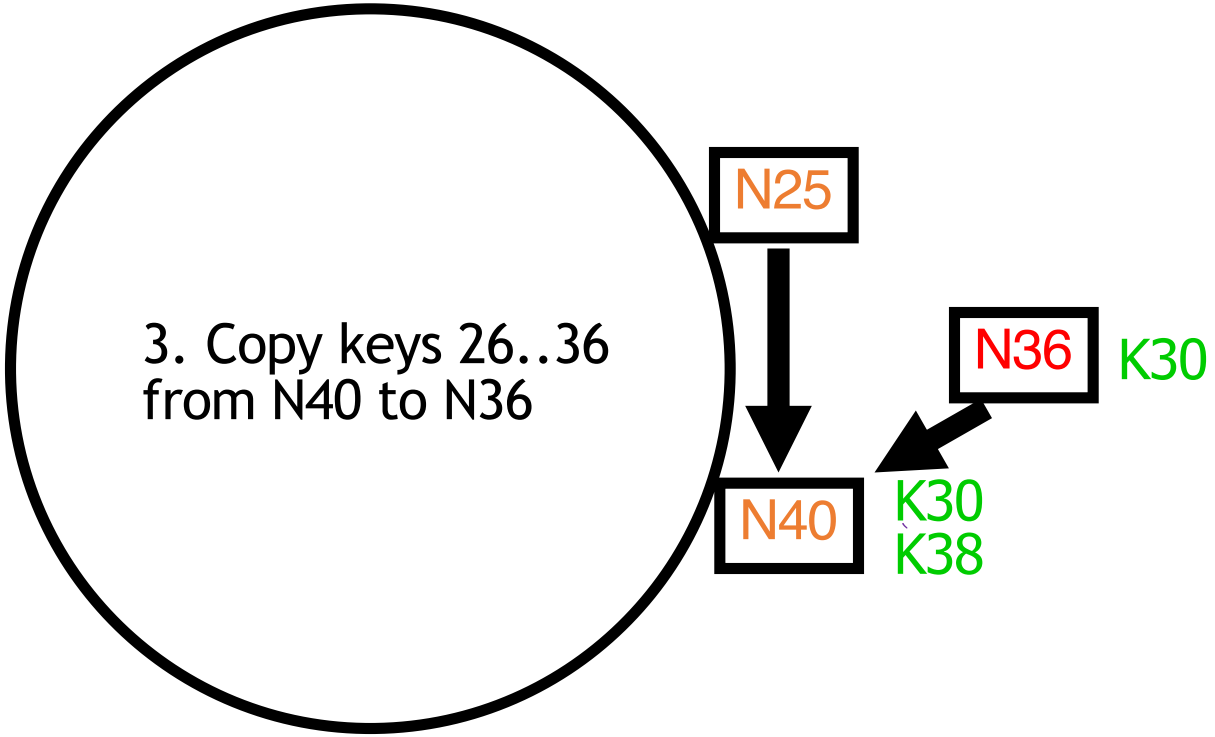
Chord Joining: linked list insert



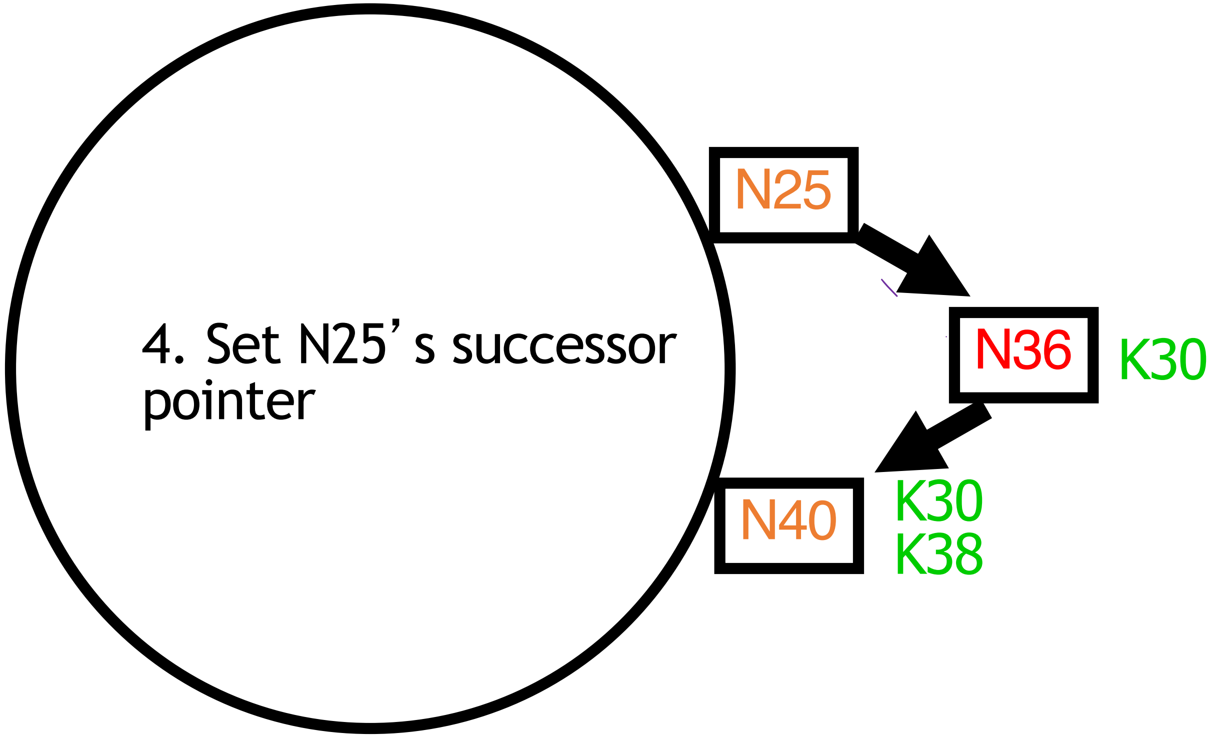
Chord Join (2)



Chord Join (3)

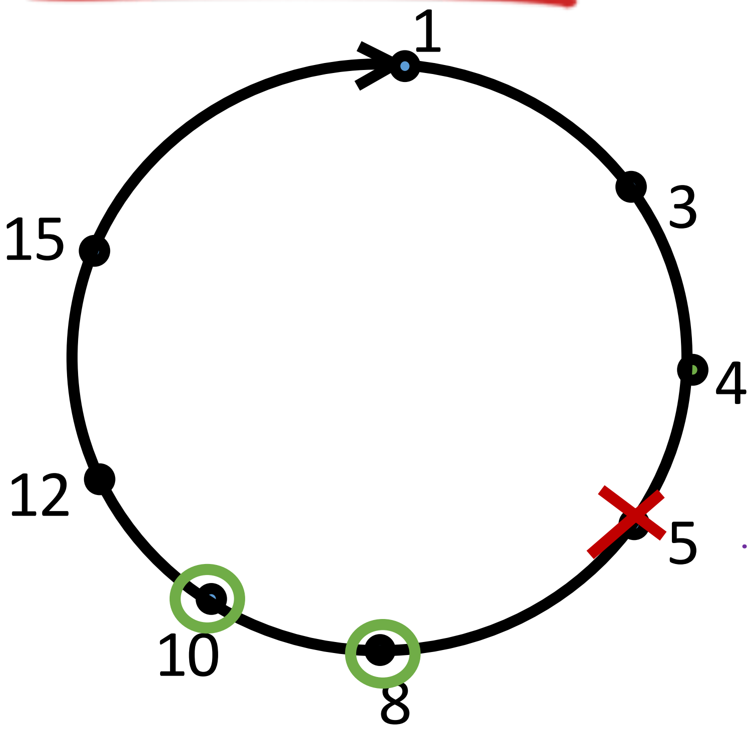


Chord Join (4)
[Done later, in stabilization]



Update other routing entries in the background
Correct successors produce correct lookups

Peer churn



handling peer churn:

- peers may come and go (churn)
- each peer knows address of its two successors
- each peer periodically pings its two successors to check aliveness
- if immediate successor leaves, choose next successor as new immediate successor

example: peer 5 abruptly leaves

- peer 4 detects peer 5's departure; makes 8 its immediate successor
- 4 asks 8 who its immediate successor is; makes 8's immediate successor its second successor.

Overview

- Basics
- Historical P2P
  - Napster
  - Gnutella
  - Kazaa
- Distributed hash tables
  - Basics
  - Chord
  - BitTorrent

Q: How to share efficiently search for and share files between peers?

BitTorrent: Chunk Request Order

- Which chunks to request?
  - Could download in order
  - Like an HTTP client does
- Problem: many peers have the early chunks
  - Peers have little to share with each other
  - Limiting the scalability of the system
- Problem: eventually nobody has rare chunks
  - E.g., the chunks need the end of the file
  - Limiting the ability to complete a download
- Possible solutions: random selection, rarest first

BitTorrent: Rarest Chunk First

- Which chunks to request first?
  - Chunk with fewest available copies (i.e., rarest chunk)

- Benefits to the peer
  - Avoid starvation when some peers depart

- Benefits to the system
  - Avoid starvation across all peers wanting a file
  - Balance load by equalizing # of copies of chunks

Free-Riding in P2P Networks

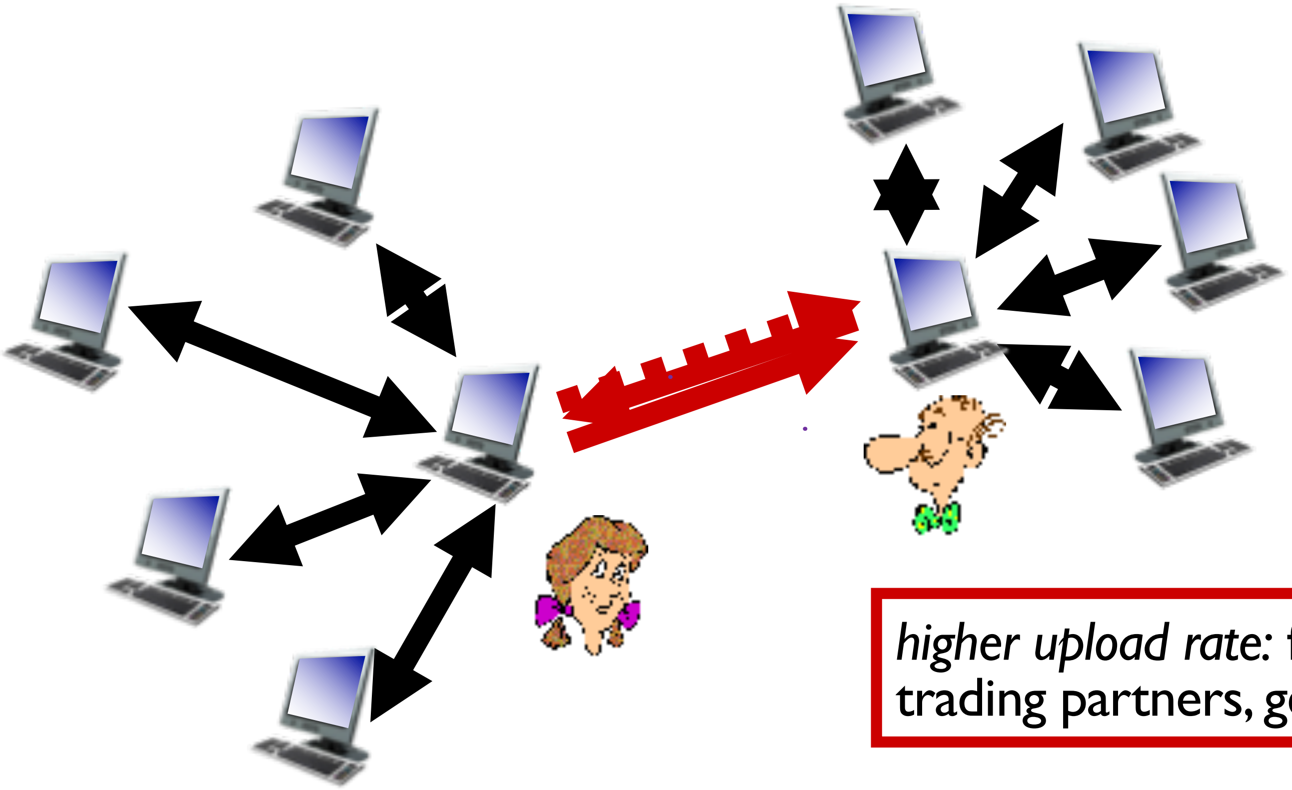
- Vast majority of users are free-riders
  - Most share no files and answer no queries
  - Others limit # of connections or upload speed
- A few "peers" essentially act as servers
  - A few individuals contributing to the public good
  - Making them hubs that basically act as a server
- BitTorrent prevents free riding
  - Allow the fastest peers to download from you
  - Occasionally let some free loaders download

Bit-Torrent: Preventing Free-Riding

- Peer has limited upload bandwidth
  - And must share it among multiple peers
  - Tit-for-tat: favor neighbors uploading at highest rate
- Rewarding the top four neighbors
  - Measure download bit rates from each neighbor
  - Reciprocate by sending to the top four peers
- Optimistic unchoking
  - Randomly try a new neighbor every 30 seconds
  - So new neighbor has a chance to be a better partner

BitTorrent: tit-for-tat

(1) Alice "optimistically unchokes" Bob
(2) Alice becomes one of Bob's top-four providers
(3) Bob reciprocates; Bob becomes one of Alice's top-four providers



higher upload rate: find better trading partners, get file faster!

BitTyrant: Gaming BitTorrent

- BitTorrent can be gamed, too
  - Peer uploads to top N peers at rate 1/N
  - E.g., if N=4 and peers upload at 15, 12, 10, 9, 8, 3
  - … peer uploading at rate 9 gets treated quite well

- Best to be the Nth peer in the list, rather than 1st
  - Offer just a bit more bandwidth than low-rate peers
  - And you'll still be treated well by others

- BitTyrant
  - Uploads at higher rates to higher-bandwidth peers

Lessons and Limitations

- Client-Server performs well
  - But not always feasible: Performance not often key issue!
- For the following, you should choose a system that's:

(A) Flood-based (B) DHT-based (C) Either (D) None

  - Scalability
  - Decentralization of visibility and liability
  - Finding popular stuff
  - Finding unpopular stuff
  - Local queries
  - Performance guarantees

Lessons and Limitations

- Client-Server performs well
  - But not always feasible: Performance not often key issue!
- For the following, you should choose a system that's:

(A) Flood-based (B) DHT-based (C) Either (D) None

  - Scalability B
  - Decentralization of visibility and liability C
  - Finding popular stuff A (C?)
  - Finding unpopular stuff B
  - Local queries A
  - Performance guarantees B

References

- Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM* 2001.
- Mohammad Alizadeh, MIT
- *Computer Networking: A Top-Down Approach* (6th ed), James Kurose and Keith Ross, 2011.