P2P file sharing

Notes based on notes by K.W. Ross, J. Kurose, D. Rubenstein, and others

Example

- Alice runs P2P client application on her notebook computer
- intermittently connects to Internet; gets new IP address for each connection
- asks for "Hey Jude"
- application displays other peers that have copy of Hey Jude.
- Alice chooses one of the peers, Bob.
- file is copied from Bob's PC to Alice's notebook: HTTP
- while Alice downloads, other users uploading from Alice.
- Alice's peer is both a Web client and a transient Web server.

All peers are servers = highly scalable!

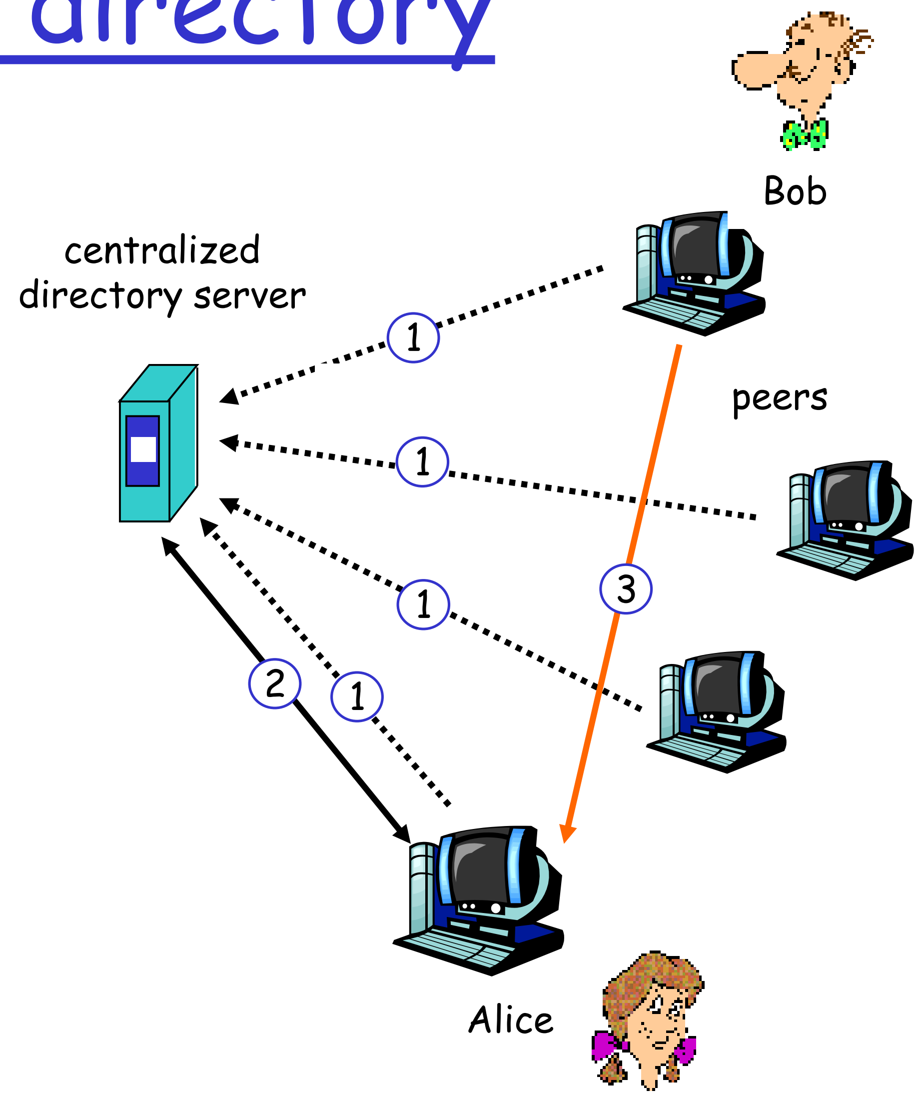
P2P: centralized directory

original "Napster" design

1) when peer connects, it informs central server:
   - IP address
   - content

2) Alice queries for "Hey Jude"

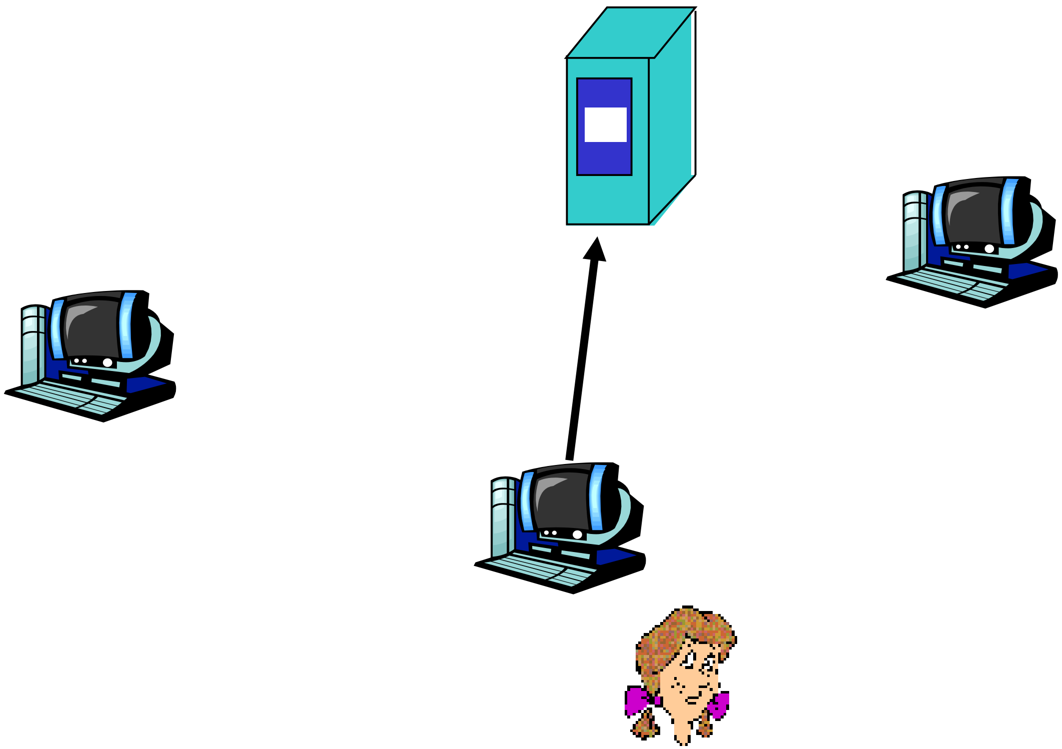
3) Alice requests file from Bob



Napster

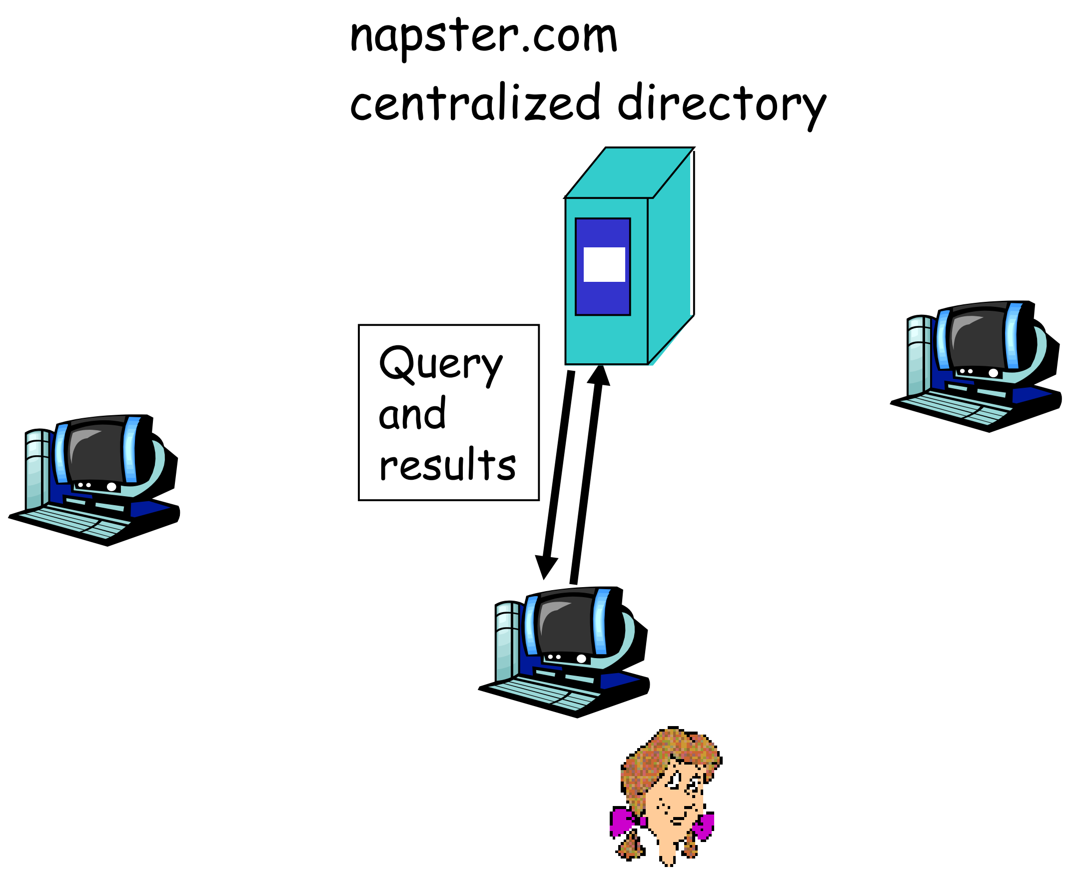
1. File list and IP address is uploaded

napster.com
centralized directory



Napster

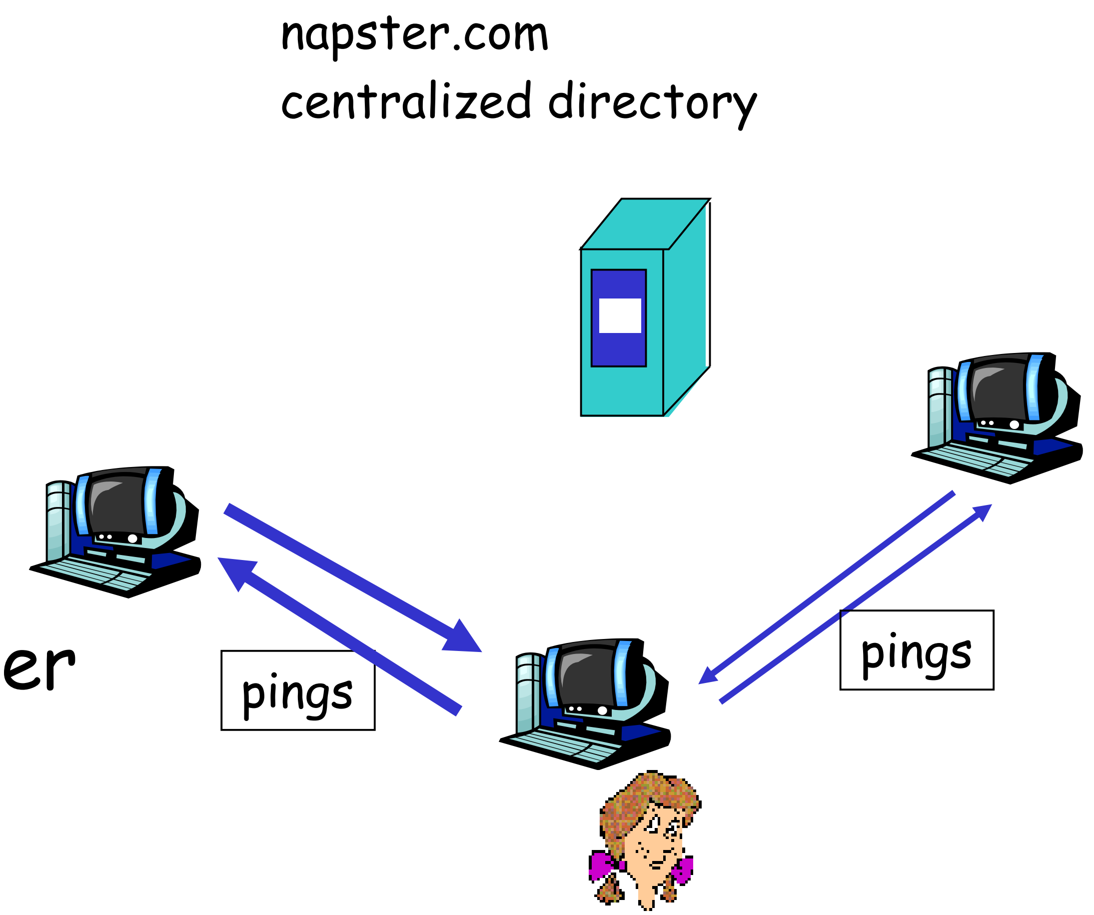
2. User requests search at server.



Napster

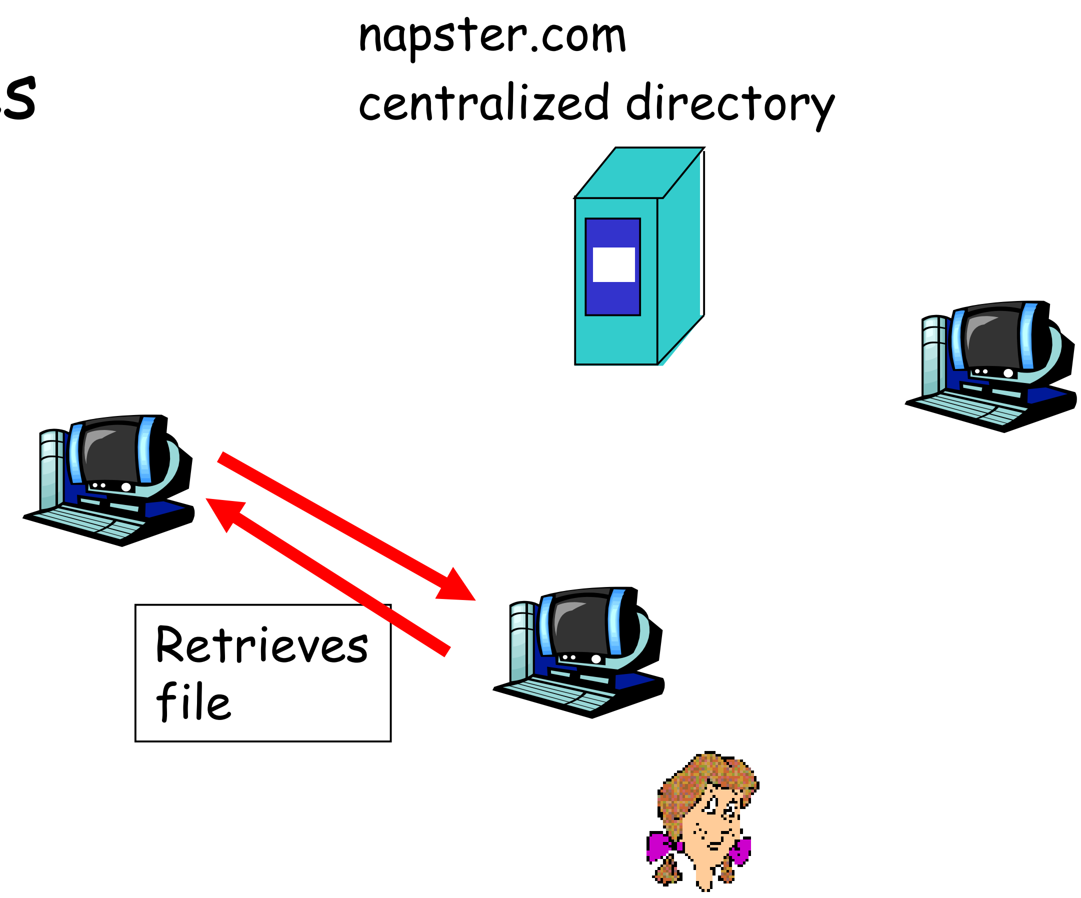
3. User pings hosts that apparently have data.

   Looks for ***best*** transfer rate.



Napster

4. User chooses server



Napster's centralized server farm had difficult time keeping up with traffic

P2P: problems with centralized directory

- single point of failure
- performance bottleneck
- copyright infringement: "target" of lawsuit is obvious

file transfer is decentralized, but locating content is highly centralized

Gnutella

- focus: decentralized method of searching for files
  - central directory server no longer the bottleneck
  - more difficult to "pull plug"
- each application instance serves to:
  - store selected files
  - route queries from and to its neighboring peers
  - respond to queries if file stored locally
  - serve files

Query flooding: Gnutella

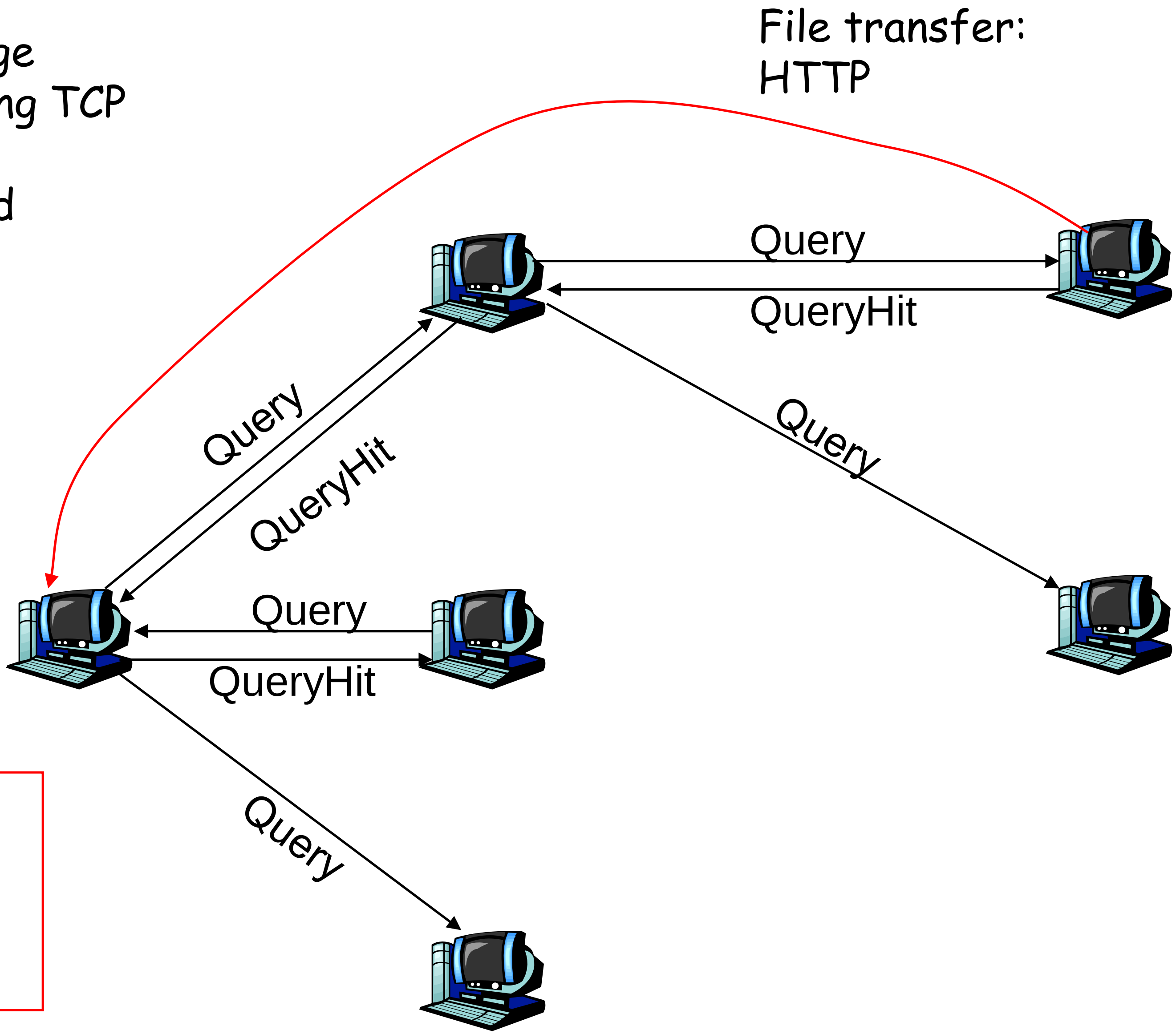
- fully distributed
  - no central server
- public domain protocol
- many Gnutella clients implementing protocol

overlay network: graph

- edge between peer X and Y if there's a TCP connection
- all active peers and edges form overlay net
- edge: virtual (*not* physical) link
- given peer typically connected with < 10 overlay neighbors

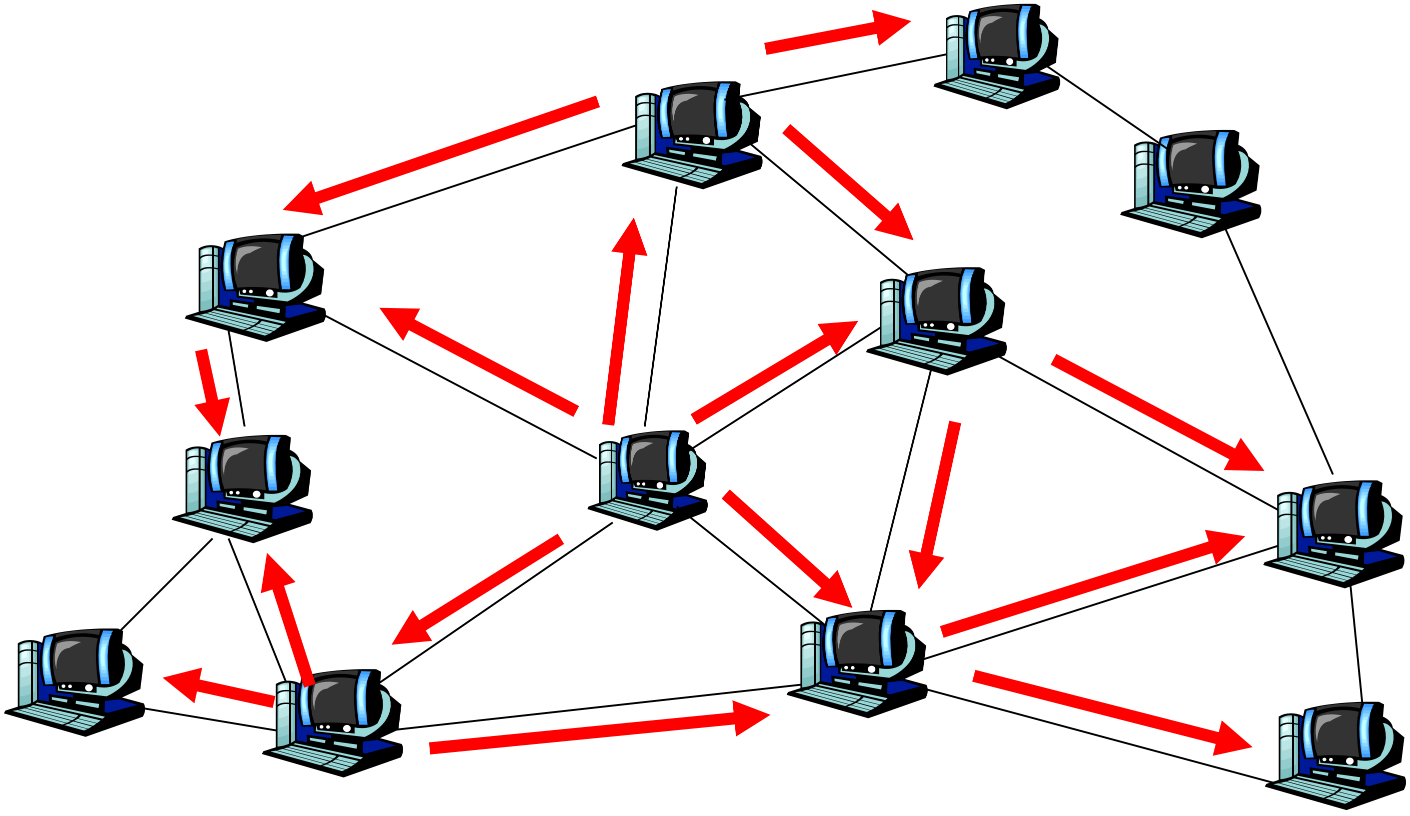
Gnutella: protocol

- ❒ Query message sent over existing TCP connections
- ❒ peers forward Query message
- ❒ QueryHit sent over reverse path

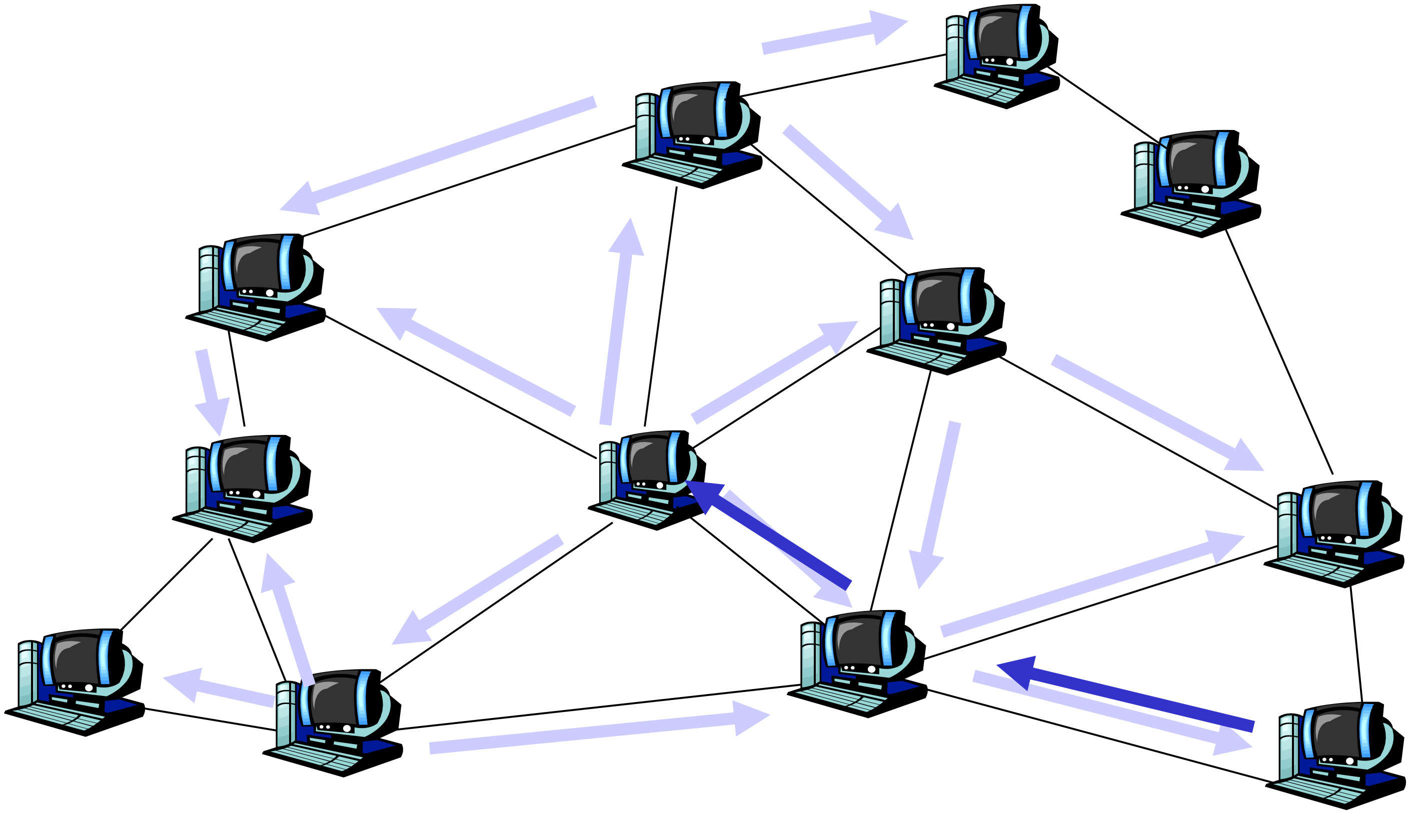


Scalability:
limited scope flooding

Distributed Search/Flooding



Distributed Search/Flooding

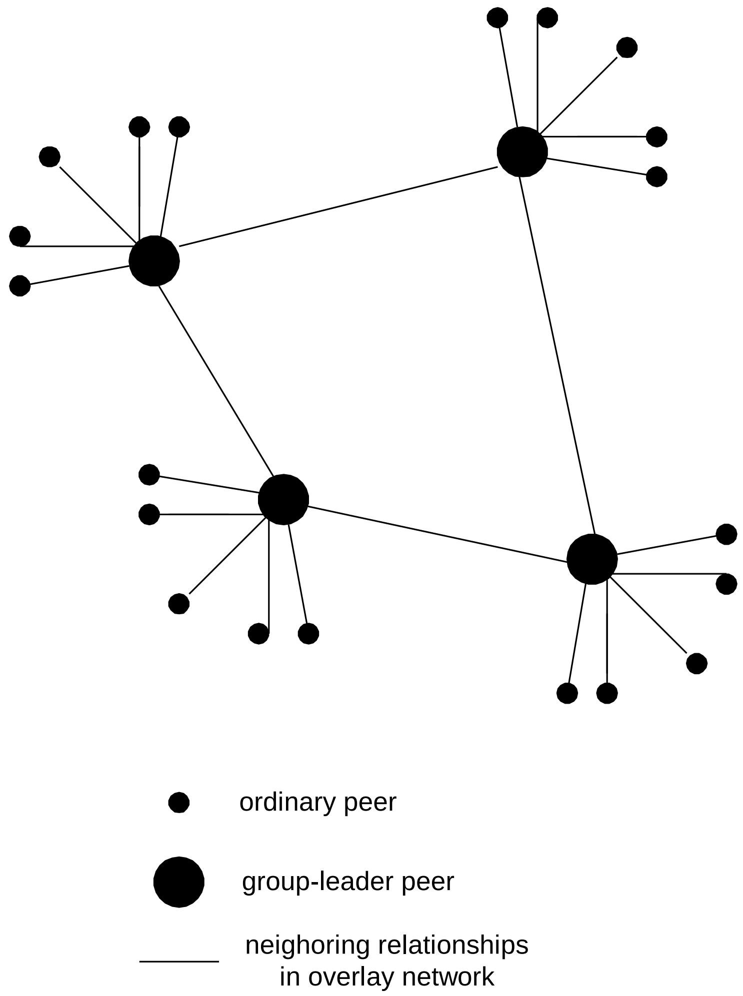


Gnutella: Peer joining

1. joining peer Alice must find another peer in Gnutella network: use list of candidate peers
2. Alice sequentially attempts TCP connections with candidate peers until connection setup with Bob
3. *Flooding:* Alice sends Ping message to Bob; Bob forwards Ping message to his overlay neighbors (who then forward to their neighbors….)
   - peers receiving Ping message respond to Alice with Pong message
4. Alice receives many Pong messages, and can then setup additional TCP connections

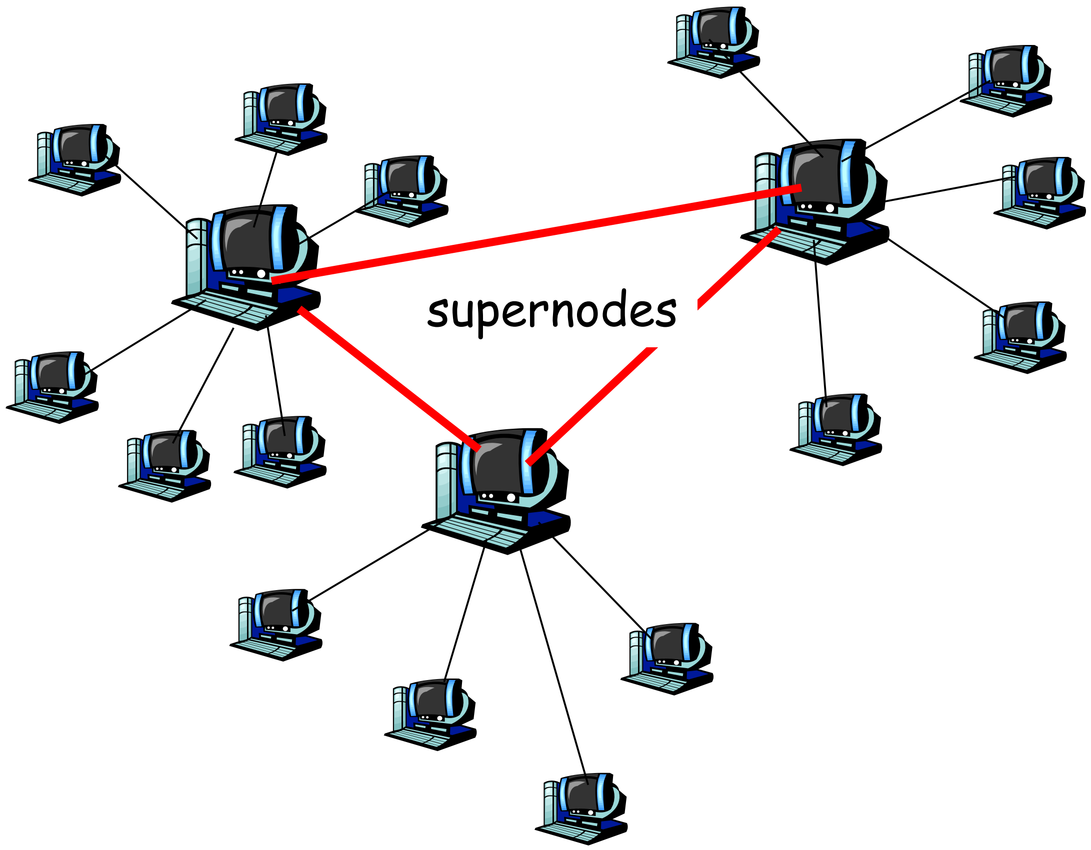
Hierarchical Overlay

- between centralized index, query flooding approaches
- each peer is either a *group leader* or assigned to a group leader.
  - TCP connection between peer and its group leader.
  - TCP connections between some pairs of group leaders.
- group leader tracks content in its children



KaZaA: Architecture

- Each peer is either a supernode or is assigned to a supernode
- Each supernode knows about many other supernodes (almost mesh overlay)



KaZaA: Architecture (2)

- Nodes that have more connection bandwidth and are more available are designated as supernodes
- Each supernode acts as a mini-Napster hub, tracking the content and IP addresses of its descendants
- Guess@peak: supernode had (on average) 200-500 descendants; roughly 10,000 supernodes
- There is also dedicated user authentication server and supernode list server

KaZaA: Overlay maintenance

- List of potential supernodes included within software download
- New peer goes through list until it finds operational supernode
  - Connects, obtains more up-to-date list
  - Node then pings 5 nodes on list and connects with the one with smallest RTT
- If supernode goes down, node obtains updated list and chooses new supernode

KaZaA Queries

- Node first sends query to supernode
  - Supernode responds with matches
  - If x matches found, done.
- Otherwise, supernode forwards query to subset of supernodes
  - If total of x matches found, done.
- Otherwise, query further forwarded
  - Probably by original supernode rather than recursively

Parallel Downloading; Recovery

- If file is found in multiple nodes, user can select parallel downloading
- Most likely HTTP byte-range header used to request different portions of the file from different nodes

- Automatic recovery when server peer stops sending file

KaZaA Corporate Structure

- Software developed by FastTrack in Amsterdam
- FastTrack also deploys KaZaA service
- FastTrack licenses software to Music City (Morpheus) and Grokster
- Later, FastTrack terminates license, leaves only KaZaA with killer 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
- Code in Estonia

Lessons learned from KaZaA

KaZaA provides powerful file search and transfer service without server infrastructure

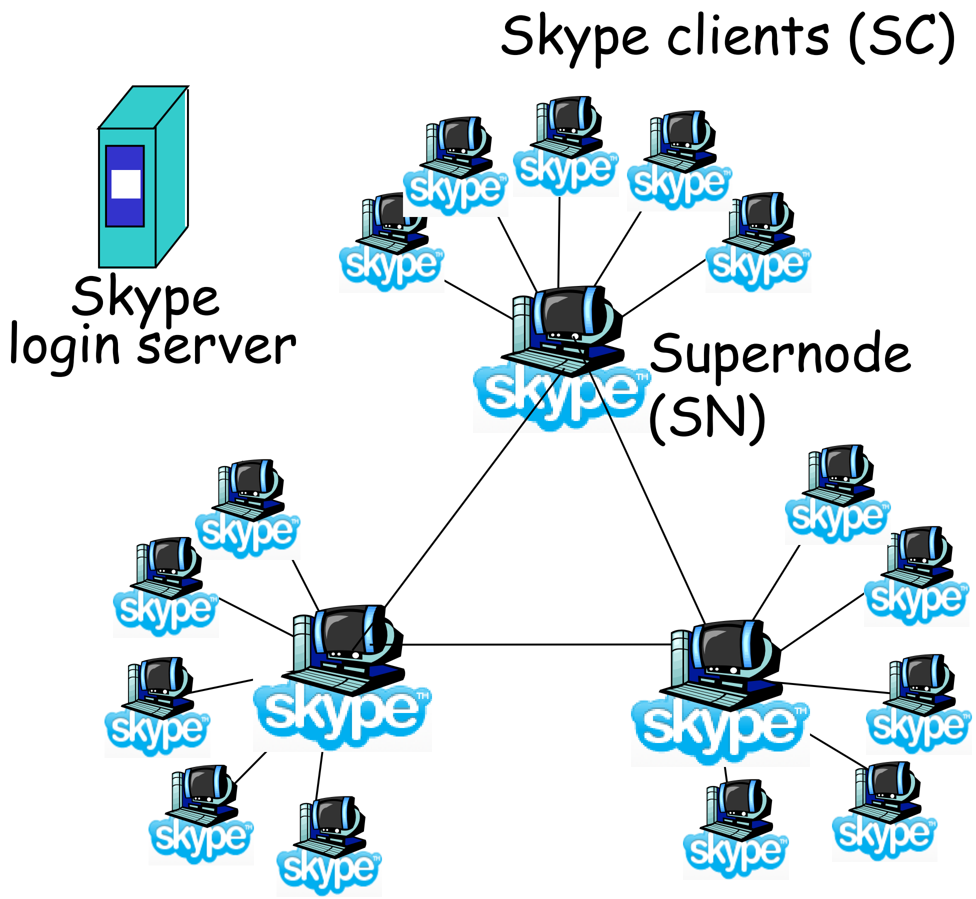
- Exploit heterogeneity
- Provide automatic recovery for interrupted downloads
- Powerful, intuitive user interface

Copyright infringement

- International cat-and-mouse game
- With distributed, serverless architecture, can the plug be pulled?
- Prosecute users?
- Launch DoS attack on supernodes?
- Pollute?

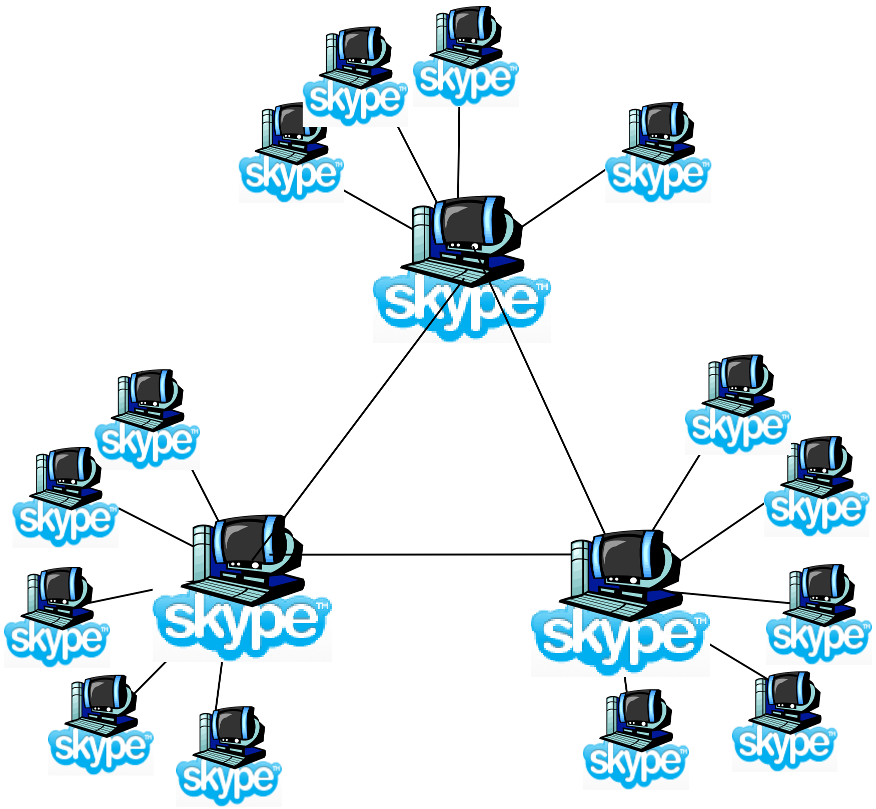
P2P Case study: Skype

- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with Supernodes (SNs)
- Index maps usernames to IP addresses; distributed over SNs



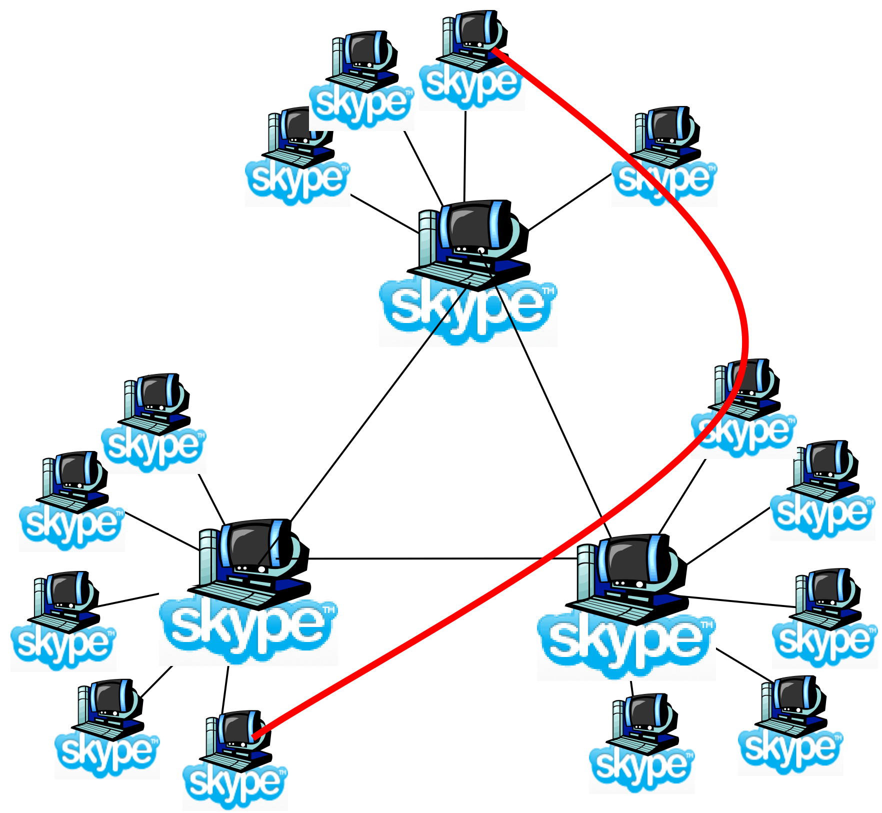
Peers as relays

- Problem when both Alice and Bob are behind "NATs".
  - NAT prevents an outside peer from initiating a call to insider peer



Peers as relays

- Problem when both Alice and Bob are behind "NATs".
  - NAT prevents an outside peer from initiating a call to insider peer
- Solution:
  - Using Alice's and Bob's SNs, Relay is chosen
  - Each peer initiates session with relay.
  - Peers can now communicate through NATs via relay



Modeling Unstructured P2P Networks

- In comparison to DHT-based searches, unstructured searches are
  - simple to build
  - simple to understand algorithmically
- Little concrete is known about their performance
- Q: what is the expected overhead of a search?
- Q: how does caching pointers help?

Replication

- Scenario
  - Nodes cache copies (or pointers to) content
    - object info can be "pushed" from nodes that have copies
    - more copies leads to shorter searches
  - Caches have limited size: can't hold everything
  - Objects have different popularities: different content requested at different rates
- Q: How should the cache be shared among the different content?
  - Favor items under heavy demand too much then lightly demanded items will drive up search costs
  - Favor a more "flat" caching (i.e., independent of popularity), then frequent searches for heavily-requested items will drive up costs
- Is there an optimal strategy?

Model

- Given
  - m objects, n nodes, each node can hold c objects, total system capacity = cn
  - q_i is the request rate for the i^{th} object, $q_1 \geq q_2 \geq \ldots \geq q_m$
  - p_i is the fraction of total system capacity used to store object i, $\sum p_i = 1$
- Then
  - Expected length of search for object i = K / p_i for some constant K
    - note: assumes search selects node w/ replacement, search stops as soon as object found
  - Network "bandwidth" used to search for all objects:

    $B = \sum q_i K / p_i$
- Goal: Find allocation for $\{p_i\}$ (as a function of $\{q_i\}$) to minimize B
- Goal 2: Find distributed method to implement this allocation of $\{p_i\}$

Some possible choices for $\{p_i\}$

- Consider some typical allocations used in practice
  - Uniform: $p_1 = p_2 = \ldots = p_m = 1/m$
    - easy to implement: whoever creates the object sends out cn/m copies
  - Proportional: $p_i = a\, q_i$ where $a = 1/\sum q_i$ is a normalization constant
    - also easy to implement: keep the received copy cached
- What is $B = \sum q_i\, K / p_i$ for these two policies?
  - Uniform: $B = \sum q_i\, K / (1/m) = Km/a$
  - Proportional: $B = \sum q_i\, K / (a\, q_i) = Km/a$
- B is the same for the Proportional and Uniform policies!

In between Proportional and Uniform

- Uniform: $p_i / p_{i+1} = 1$, Proportional: $p_i / p_{i+1} = q_i / q_{i+1} \geq 1$
- In between: $1 \leq p_i / p_{i+1} \leq q_i / q_{i+1}$
- Claim: any in-between allocation has lower B than B for Uniform / Proportional
- Proof: Omitted here
- Consider Square-Root allocation: $p_i = \mathrm{sqrt}(q_i) / \sum \mathrm{sqrt}(q_i)$
- Thm: Square-Root is optimal
- Proof (sketch):
  - Noting $p_m = 1 - (p_1 + \ldots + p_{m-1})$
  - write $B = F(p_1, \ldots, p_{m-1}) = \sum^{m-1} q_i/p_i + q_m/(1- \sum^{m-1} p_i)$
  - Solving $dF/dp_i = 0$ gives $p_i = p_m \, \mathrm{sqrt}(q_i/q_m)$

Distributed Method for Square-Root Allocation

- Assumption: each copy in the cache disappears from the cache at some rate independent of the object cached (e.g., object lifetime is i.i.d.)
- Algorithm Sqrt-Cache: cache a copy of object i (once found) at each node visited while searching for object i
- Claim Algorithm implements Square-Root Allocation

Proof of Claim

- Sketch of Proof of Correctness:
  - Let $f_i(t)$ be fraction of locations holding object i @ time t
  - $p_i = \lim_{t\to\infty} f_i(t)$
  - At time t, using Sqrt-Cache, object i populates cache at avg rate $r_i = q_i / f_i(t)$
  - When $f_i(t) / f_j(t) < sqrt(q_i) / sqrt(q_j)$, then
    - $r_i(t) / r_j(t) = q_i f_j(t) / q_j f_i(t) > sqrt(q_i) / sqrt(q_j)$
    - hence, ratio $f_i(t) / f_j(t)$ will increase
  - When $f_i(t) / f_j(t) > sqrt(q_i) / sqrt(q_j)$, then
    - $r_i(t) / r_j(t) = q_i f_j(t) / q_j f_i(t) < sqrt(q_i) / sqrt(q_j)$
    - hence, ratio $f_i(t) / f_j(t)$ will decrease
  - Steady state is therefore when $f_i(t) / f_j(t) = sqrt(q_i) / sqrt(q_j)$,

6. P2P Graph Structure

- What are “good” P2P graphs and how are they built?
- Graphs we will consider
  - Random (Erdos-Renyi)
  - Small-World
  - Scale-free

"Good" Unstructured P2P Graphs

- Desirable properties
  - each node has small to moderate degree
  - expected # of hops needed to go from a node u to a node v is small
  - easy to figure out how to find the right path
  - difficult to attack the graph (e.g., by knocking out nodes)
  - don't need extensive modifications when nodes join/leave (e.g., like in Chord, CAN, Pastry)
- Challenge: Difficult to enforce structure

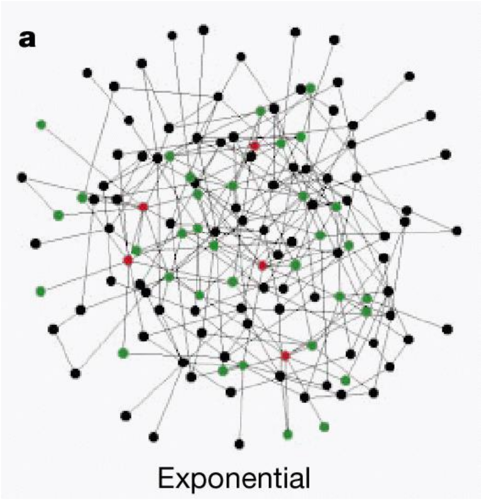
Random (Erdos-Renyi) Graphs

- For all nodes u,v, edge (u,v) is added with fixed probability p
- Performance in P2P Context: In some sense, these graphs are too random
  - long distance between pairs of nodes likely
  - difficult to build a good distributed algorithm that can find a short route between arbitrary pair of nodes

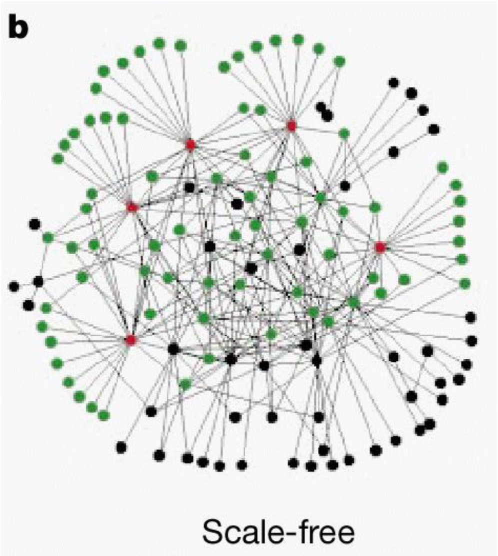
Small World Model

- Nodes have positions (e.g., on a 2D graph)
- Let d(u,v) be the distance between nodes u & v
- Constants p, q, r chosen:
  - each node u connects to all other nodes v where d(u,v) < p
  - each node connects to q additional (far away) nodes drawn from distribution where edge (u,v) is selected with probability proportional to $d(u,v)^{-r}$
  - Each node knows all neighbors within distance p and also knows q far neighbors
  - Search method: choose the neighbor that is closest (in distance) to the desired destination

Scale-Free Graphs



- Erdos-Renyi and Small-World graphs are exponential: the degree of nodes in the network decays exponentially
- Scale-free graph: node connects to node with current degree k with probability proportional to k
  - nodes with many neighbors more likely to get more neighbors
- Scale-free graphs' degree decays according to a power law: Pr(node has k neighbors) = $k^{-\alpha}$



Are Scale-Free Networks Better?

- Average diameter lower in Scale-Free than in Exponential graphs
- What if nodes are removed?
  - at random: scale free keeps lower diameter
  - by knowledgable attacker: (nodes of highest degree removed first): scale-free diameter grows quickly
- Same results apply using sampled Internet and WWW graphs (that happen to be scale-free)