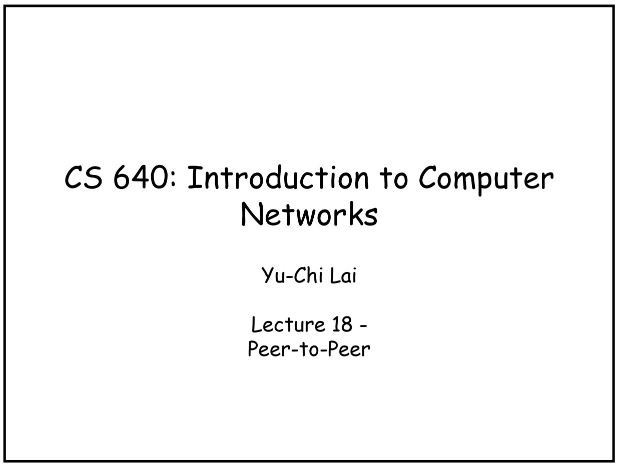
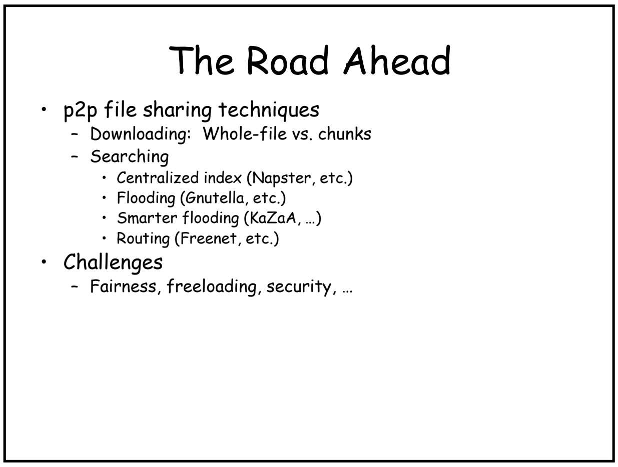


CS 640: Introduction to Computer Networks

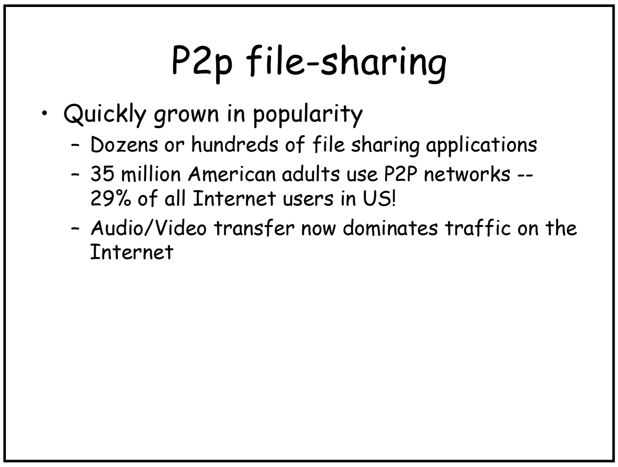
Yu-Chi Lai

Lecture 18 -
Peer-to-Peer



The Road Ahead

- p2p file sharing techniques
  - Downloading: Whole-file vs. chunks
  - Searching
    - Centralized index (Napster, etc.)
    - Flooding (Gnutella, etc.)
    - Smarter flooding (KaZaA, …)
    - Routing (Freenet, etc.)
- Challenges
  - Fairness, freeloading, security, …



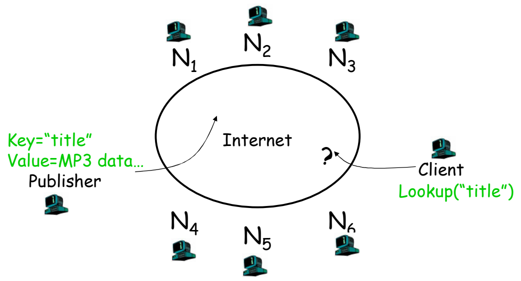
P2p file-sharing

- Quickly grown in popularity
  - Dozens or hundreds of file sharing applications
  - 35 million American adults use P2P networks -- 29% of all Internet users in US!
  - Audio/Video transfer now dominates traffic on the Internet

What's out there?

	Central	Flood	Super-node flood	Route
Whole File	Napster	Gnutella		Freenet
Chunk Based	BitTorrent		KaZaA (bytes, not chunks)	DHTs eDonkey 2000

Publishing/Searching



N_1 N_2 N_3

Key="title"
Value=MP3 data...
Publisher

Internet

?

Client
Lookup("title")

N_4 N_5 N_6

Searching

- Needles vs. Haystacks
  - Searching for top 40, or an obscure punk track from 1981 that nobody's heard of?
- Search expressiveness
  - Whole word? Regular expressions? File names? Attributes? Whole-text search?
    - (e.g., p2p gnutella or p2p google?)

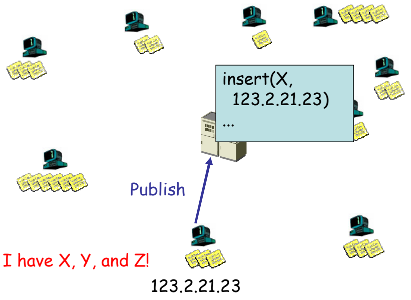
Framework

- Common Primitives:
  - **Join**: how to I begin participating?
  - **Publish**: how do I advertise my file?
  - **Search**: how to I find a file?
  - **Fetch**: how to I retrieve a file?

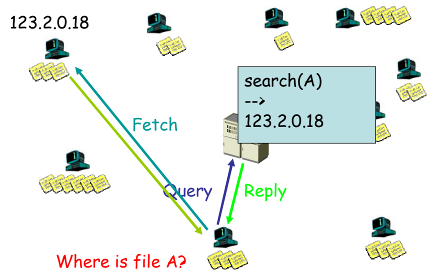
Napster: Overview

- History
  - 1999: Sean Fanning launches Napster
  - Peaked at 1.5 million simultaneous users
  - Jul 2001: Napster shuts down
- 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: Publish



Napster: Search



123.2.0.18

search(A)
-->
123.2.0.18

Fetch

Query

Reply

Where is file A?

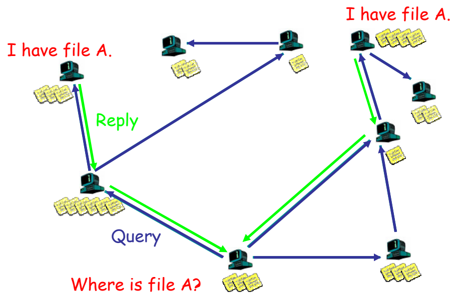
Napster: Discussion

- Pros:
  - Simple
  - Search scope is O(1)
  - Controllable (pro or con?)
- Cons:
  - Server maintains lot of state
  - Server does all processing
  - Single point of failure

Gnutella: Overview

- History:
  - In 2000, J. Frankel and T. Pepper from Nullsoft released Gnutella
  - Soon many other clients: Bearshare, Morpheus, LimeWire…
  - In 2001, many protocol enhancements including "ultrapeers"
- Query Flooding:
  - **Join**: on startup, client contacts a few other nodes; these become its "neighbors"
    - Ping-Pong protocol
  - **Publish**: no need
  - **Search**: ask neighbors, who ask their neighbors, and so on... when/if found, reply to sender.
    - TTL limits propagation
  - **Fetch**: get the file directly from peer

Gnutella: Search

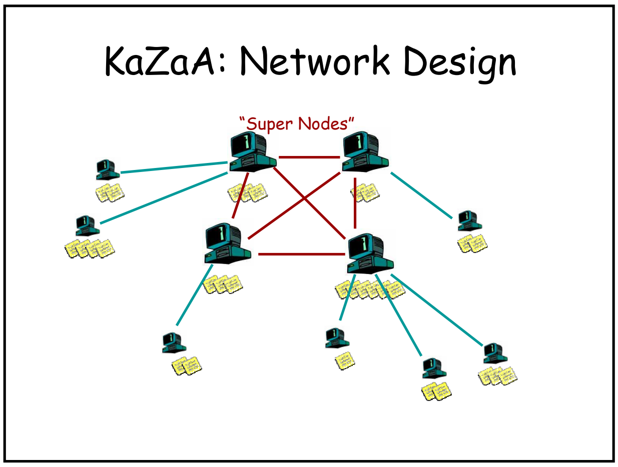


Gnutella: Discussion

- Pros:
  - Fully de-centralized
  - Search cost distributed
  - Processing @ each node permits powerful search semantics
- Cons:
  - Search scope is O(*N*)
  - Search time is O(???)
  - Nodes leave often, network unstable
- TTL-limited search works well for haystacks.
  - For scalability, does NOT search every node. May have to re-issue query later

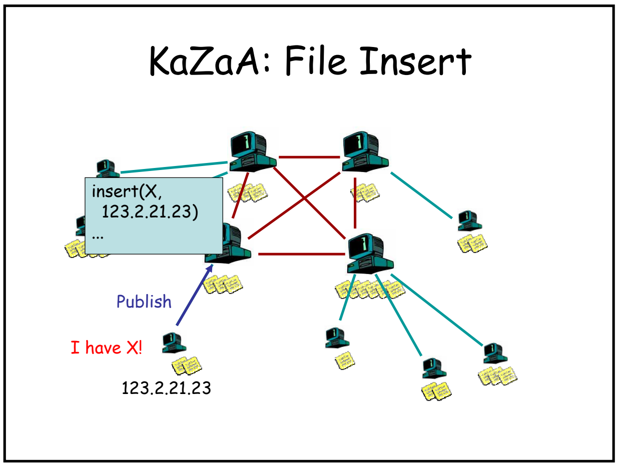
KaZaA: Overview

- Gnutella X Napster
  - No didicated server
  - But.. not all peers are equal!
- “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: Network Design

"Super Nodes"



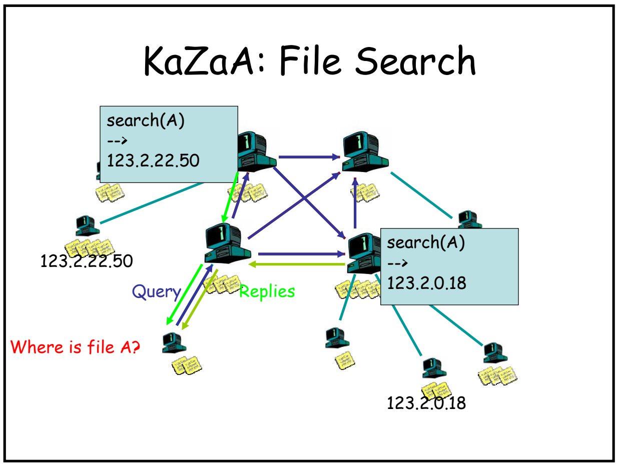
KaZaA: File Insert

insert(X,
123.2.21.23)
...

Publish

I have X!

123.2.21.23



KaZaA: File Search

search(A)
-->
123.2.22.50

123.2.22.50

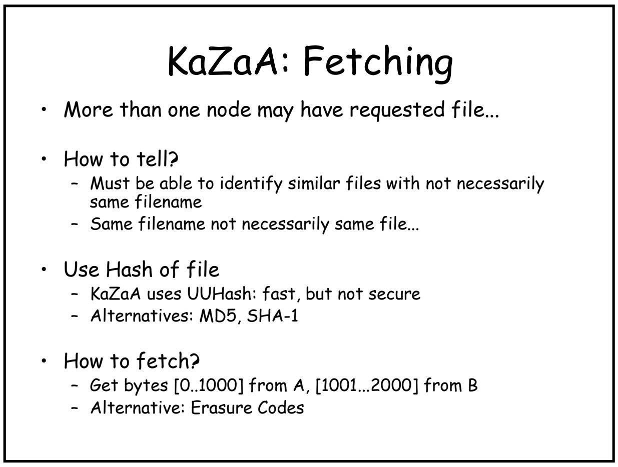
Query

Replies

Where is file A?

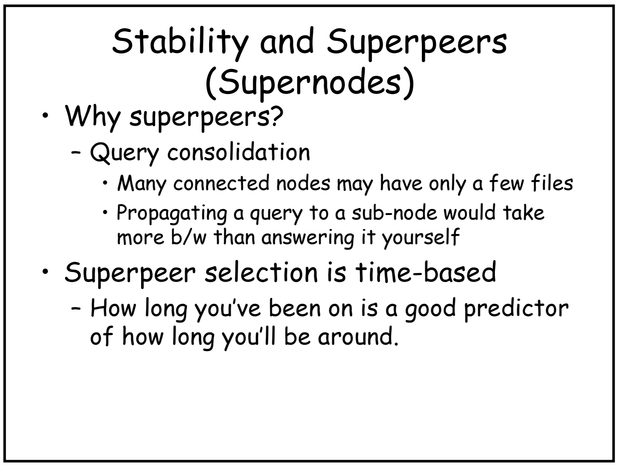
search(A)
-->
123.2.0.18

123.2.0.18



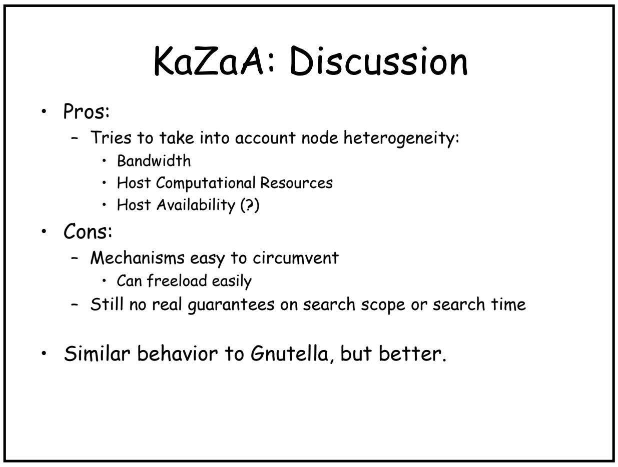
KaZaA: Fetching

- More than one node may have requested file...
- How to tell?
  - Must be able to identify similar files with 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



Stability and Superpeers (Supernodes)

- Why superpeers?
  - Query consolidation
    - Many connected nodes may have only a few files
    - Propagating a query to a sub-node would take more b/w than answering it yourself
- Superpeer selection is time-based
  - How long you've been on is a good predictor of how long you'll be around.



KaZaA: Discussion

- Pros:
  - Tries to take into account node heterogeneity:
    - Bandwidth
    - Host Computational Resources
    - Host Availability (?)
- Cons:
  - Mechanisms easy to circumvent
    - Can freeload easily
  - Still no real guarantees on search scope or search time
- Similar behavior to Gnutella, but better.

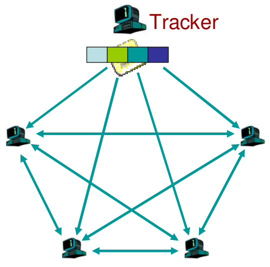
BitTorrent: History

- Key Motivation:
  - Popularity exhibits temporal locality (Flash Crowds)
  - E.g., Slashdot effect, CNN on 9/11, new movie/game release
- Focused on Efficient *Fetching*, not *Searching*:
  - Distribute the *same* file to all peers
  - Single publisher, multiple downloaders
- Has some "real" publishers

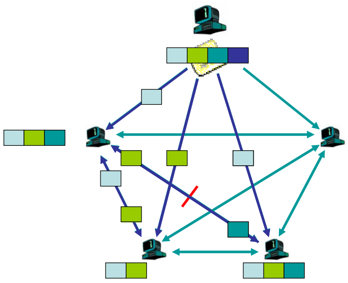
BitTorrent: Overview

- Swarming:
  - **Publish**: Run a tracker server.
  - **Search**: Out-of-band. E.g., use Google to find a tracker for the file you want.
  - **Join**: contact centralized "tracker" server, get a list of peers.
  - **Fetch**: Download chunks of the file from your peers. Upload chunks you have to them.
- Big differences from Napster:
  - Chunk based downloading
  - "few large files" focus
  - Anti-freeloading mechanisms

BitTorrent: Publish/Join



BitTorrent: Fetch



BitTorrent: Sharing Strategy

- Employ "Tit-for-tat" sharing strategy
  - A is downloading from some other people
    - A will let the fastest N of those download from him
  - Be optimistic: occasionally let freeloaders download
    - Otherwise no one would ever start!
    - Also allows you to discover better peers to download from when they reciprocate

BitTorrent: Discussion

- Pros:
  - Works reasonably well in practice
  - Gives peers incentive to share resources; avoids freeloaders
- Cons:
  - Pareto Efficiency relatively weak condition
  - Central tracker server needed to bootstrap swarm
  - (Tracker is a design choice, not a requirement, as you know from your projects. Could easily combine with other approaches.)

Distrubuted Hash Table: Overview

- Decentralized Routing:
  - **Join**: locate at the nearest hashing area by key
  - **Publish**: use the key to insert the document
  - **Search**: use the key to look up the location
  - **Fetch**: the terminal node sends a reply along the route specified by the intermediate nodes' records of pending requests

Locate An Object

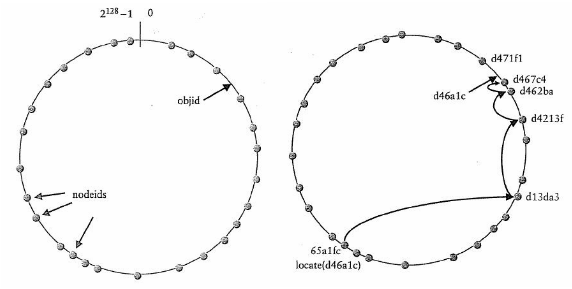
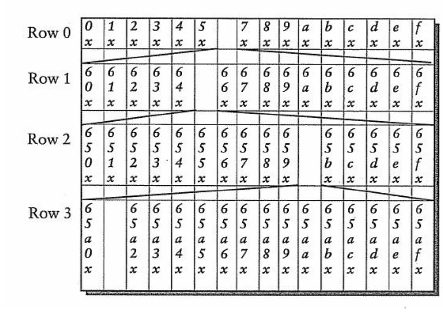
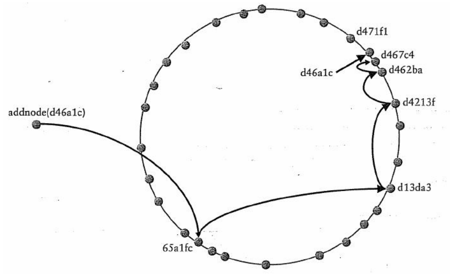


Table in A Node



Row 0	0x	1x	2x	3x	4x	5x		7x	8x	9x	ax	bx	cx	dx	ex	fx
Row 1	60x	61x	62x	63x	64x		66x	67x	68x	69x	6ax	6bx	6cx	6dx	6ex	6fx
Row 2	650x	651x	652x	653x	654x	655x	656x	657x	658x	659x		65bx	65cx	65dx	65ex	65fx
Row 3	65a0x		65a2x	65a3x	65a4x	65a5x	65a6x	65a7x	65a8x	65a9x	65aax	65abx	65acx	65adx	65aex	65afx

Add A New Node



DHT: Discussion

- Pros:
  - Self-organize into a distributed, clustered structure where nodes tend to hold data items that are close together in key space
  - All node communications are identical
  - Simple data structure
- Cons:
  - No guarantee on finding the piece of data

P2P: Summary

- Many different styles; remember pros and cons of each
  - centralized, flooding, swarming, unstructured and structured routing
- Lessons learned:
  - Single points of failure are very bad
  - Flooding messages to everyone is bad
  - Underlying network topology is important
  - Not all nodes are equal
  - Need incentives to discourage freeloading
  - Privacy and security are important
  - Structure can provide theoretical bounds and guarantees