15-441 15-641 Computer Networking

Lecture 16: Delivering Content
Web and Peer-Peer
Peter Steenkiste

Fall 2014
www.cs.cmu.edu/~prs/15-441-F14

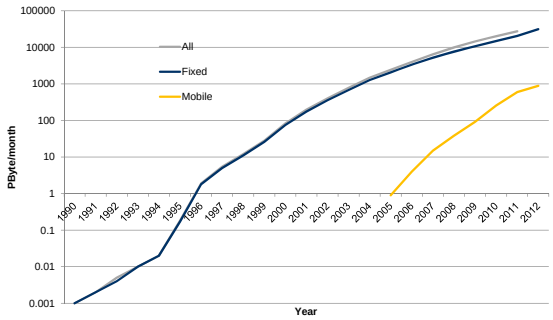
Overview

- Web
  - Protocol interactions
  - Caching
  - Cookies
- Consistent hashing
- Peer-to-peer
- CDN
- Video

Web history

- 1945: Vannevar Bush, “As we may think”, Atlantic Monthly, July, 1945.
  - Describes the idea of a distributed hypertext system.
  - A “memex” that mimics the “web of trails” in our minds.
- 1989: Tim Berners-Lee (CERN) writes internal proposal to develop a distributed hypertext system
  - Connects “a web of notes with links”.
  - Intended to help CERN physicists in large projects share and manage information
- 1990: TBL writes graphical browser for Next machines
- 1992-1994: NCSA/Mosaic/Netscape browser release

Internet Traffic History



Typical Workload (Web Pages)

- Multiple (typically small) objects per page
- File sizes
  - Heavy-tailed
    - Pareto distribution for tail
    - Lognormal for body of distribution
- Embedded references
  - Number of embedded objects also Pareto
    $Pr(X>x) = (x/x_m)^{-k}$
- This plays havoc with performance. Why?
- Solutions?

- Lots of small objects means & TCP
  - 3-way handshake
  - Lots of slow starts
  - Extra connection state

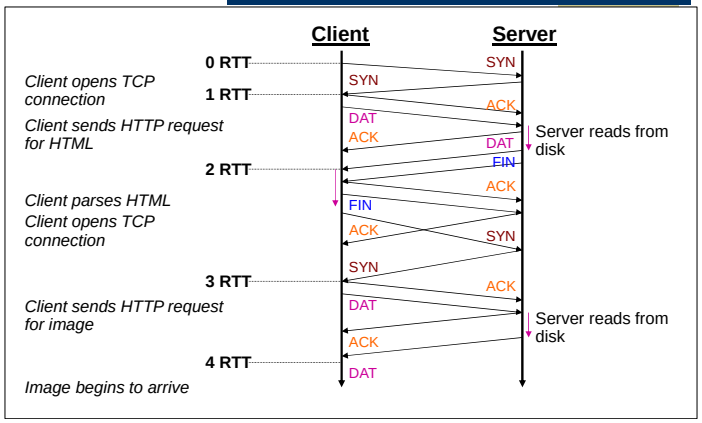
5

HTTP 0.9/1.0

- One request/response per TCP connection
  - Simple to implement
- Short transfers are very hard on TCP
  - Multiple connection setups → three-way handshake each time
    - Several extra round trips added to transfer
  - Many slow starts – low throughput because of small window
    - Never leave slow start for short transfers
  - Loss recovery is poor when windows are small
  - Lots of extra connections
    - Increases server state/processing

6

Single Transfer Example



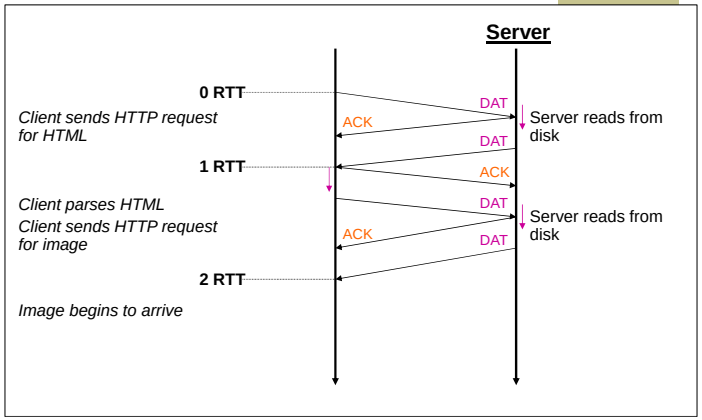
7

Improving Performance

- Multiple concurrent connections (Netscape)
  - Benefits are mixed: more state, timeouts, …
- Multiplex multiple transfers onto one TCP connection (HTTP 1.1)
  - Also allow pipelined transfers, i.e., multiple outstanding requests
- How to identify requests/responses
  - Delimiter → Server must examine response for delimiter string
  - Content-length and delimiter → Must know size of transfer in advance
  - Block-based transmission → send in multiple length delimited blocks
  - Store-and-forward → wait for entire response and then use content-length
  - Solution → use existing methods and close connection otherwise

8

Persistent Connection Solution



9

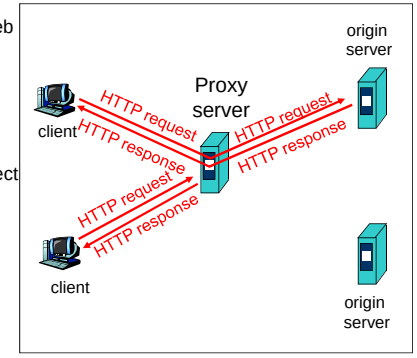
Other Problems

- Serialized transmission but first bytes may be most useful
  - May be better to get the 1st 1/4 of all images than one complete image (e.g., progressive JPEG)
  - Can "packetize" transfer over TCP, e.g., range requests
- Application specific solution to transport protocol problems. :(
  - Could fix TCP so it works well with multiple simultaneous connections
    - More difficult to deploy
- Persistent connections can significantly increase state on the server
  - Allow server to close HTTP session
  - Client can no longer send requests

10

Web Proxy Caches

- User configures browser: Web accesses via cache
- Browser sends all HTTP requests to cache
  - Object in cache: cache returns object
  - Else cache requests object from origin server, then returns object to client



11

No Caching Example (1)

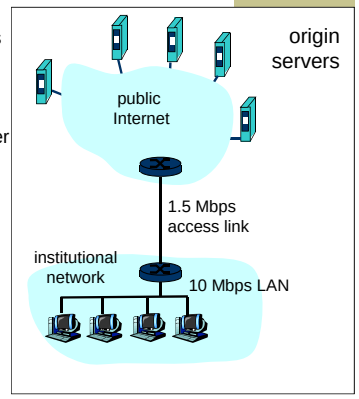
Assumptions

- Average object size = 100,000 bits
- Avg. request rate from institution's browser to origin servers = 15/sec
- Delay from institutional router to any origin server and back to router = 2 sec

Consequences

- Utilization on LAN = 15%
- Utilization on access link = 100%
- Total delay = Internet delay + access delay + LAN delay

= 2 sec + minutes + milliseconds



12

No Caching Example (2)

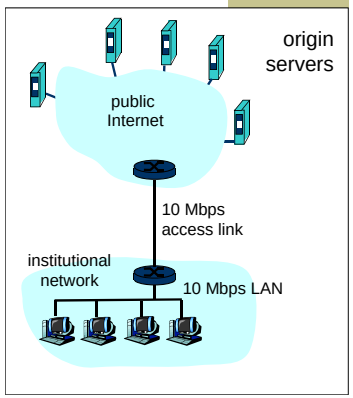
Possible solution

- Increase bandwidth of access link to, say, 10 Mbps
- Often a costly upgrade

Consequences

- Utilization on LAN = 15%
- Utilization on access link = 15%
- Total delay = Internet delay + access delay + LAN delay

= 2 sec + msecs + msecs



13

With Caching Example (3)

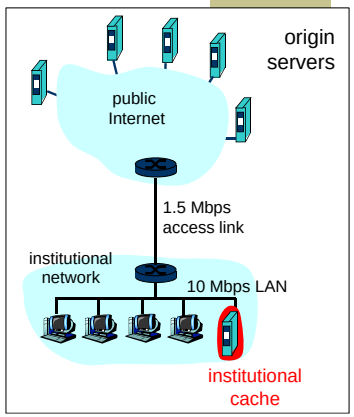
Install cache

- Suppose hit rate is .4

Consequence

- 40% requests will be satisfied almost immediately (say 10 msec)
- 60% requests satisfied by origin server
- Utilization of access link reduced to 60%, resulting in negligible delays
- Weighted average of delays

= .6*2 sec + .4*10msecs < 1.3 secs



14

HTTP Caching

- Clients often cache documents
  - Challenge: update of documents
  - If-Modified-Since requests to check
    - HTTP 0.9/1.0 used just date
    - HTTP 1.1 has an opaque "entity tag" (could be a file signature, etc.) as well
- When/how often should the original be checked for changes?
  - Check every time?
  - Check each session? Day? Etc?
  - Use Expires header
    - If no Expires, often use Last-Modified as estimate

15

Problems

- Fraction of HTTP objects that are cacheable is dropping
  - Why?
  - Major exception?
- This problem will not go away
  - Dynamic data → stock prices, scores, web cams
  - CGI scripts → results based on passed parameters
- Other less obvious examples
  - SSL → encrypted data is not cacheable
    - Most web clients don't handle mixed pages well →many generic objects transferred with SSL
  - Cookies → results may be based on past data
  - Hit metering → owner wants to measure # of hits for revenue, etc.
- What will be the end result?

16

Cookies: Keeping “state”

Many major Web sites use cookies

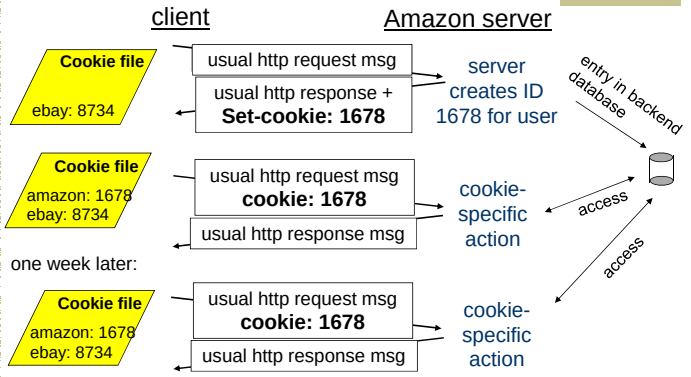
Four components:

1) Cookie header line in the HTTP response message
2) Cookie header line in HTTP request message
3) Cookie file kept on user’s host and managed by user’s browser
4) Back-end database at Web site

Example:

- Susan accesses Internet always from the same PC
- She visits a specific e-commerce site for the first time
- When initial HTTP requests arrives at the site, the site creates a unique ID and creates an entry in a backend database for that ID

Cookies: Keeping “State”



Overview

- Web
- Consistent hashing
- Peer-to-peer
- CDN
- Video

Distributing Load across Servers

- Given document XYZ, we need to choose a server to use
  - E.g., in a data center
- Suppose we use simple hashing: modulo of a hash of the name of the document
- Number servers from 1…n
  - Place document XYZ on server (XYZ mod n)
  - What happens when a servers fails? n → n-1
    - Same if different people have different measures of n
  - Why might this be bad?

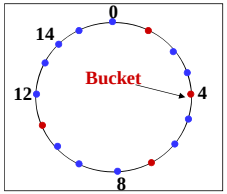
Consistent Hash: Goals

- "view" = subset of all hash buckets that are candidate locations
  - Correspond to a real server
- Desired features
  - Load – all hash buckets have a similar number of objects assigned to them
  - Smoothness – little impact on hash bucket contents when buckets are added/removed
  - Spread – small set of hash buckets that may hold an object regardless of views

21

Consistent Hash – Example

- Construction
  - Assign each of C hash buckets to random points on mod 2^n circle, where, hash key size = *n*.
  - Map object to random position on unit interval
  - Hash of object = closest bucket



- Monotone → addition of bucket does not cause movement between existing buckets
- Spread & Load → small set of buckets that lie near object
- Balance → no bucket is responsible for large number of objects

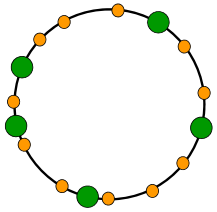
22

Consistent Hashing: Ring

- Use consistent has to map both keys and nodes to an m-bit identifier in the same (metric) identifier space
  - For example, use SHA-1 hashes
  - **Node identifier:** SHA-1 hash of IP address

    IP="198.10.10.1" —SHA-1→ ID=123
  - **Key identifier:** SHA-1 hash of key

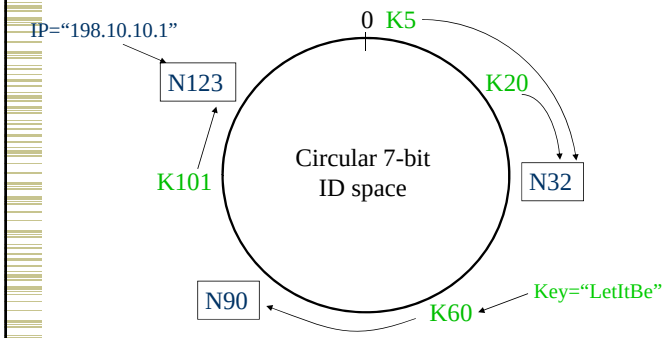
    Key="LetItBe" —SHA-1→ ID=60
- Also need "rule" for assigning keys to nodes
  - For example: "closest", higher, lower, ..



23

Consistent Hashing Example

Rule: A key is stored at its successor: node with next higher or equal ID



24

Consistent Hashing Properties

- **Load balance:** all nodes receive roughly the same number of keys
  - For *N* nodes and *K* keys, with high probability
    - Each node holds at most $(1+\varepsilon)K/N$ keys
    - Provided that K is large compared to N
- When server is added, it receives its initial work load from "neighbors" on the ring
  - "Local" operation: no other servers are affected
  - Similar property when a server is removed

25

Finer Grain Load Balancing

- Redirector knows all server IDs
- It can also track approximate "load" for more precise load balancing
  - Need to define load and be able to track it
- To balance load:
  - W_i = Hash(URL, ip of s_i) for all i
  - Sort W_i from high to low
  - Find first server with low enough load
- Benefits and drawbacks?

26

Consistent Hashing Used in Many Contexts

- Distribute load across servers in a data center
  - The redirector sits in data center
- Finding storage cluster for an object in a CDN uses centralized knowledge
  - Why?
  - Can use consistent hashing in the cluster
- Consistent hashing can also be used in a distributed setting
  - P2P systems can use it find files

27

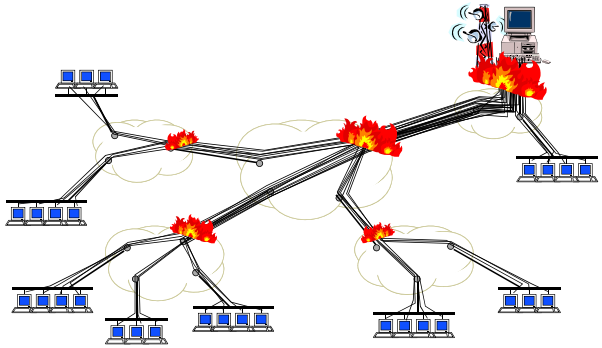
Overview

- Web
- Consistent hashing
- Peer-to-peer
  - Motivation
  - Architectures
  - TOR
  - Skype
- CDN
- Video

28

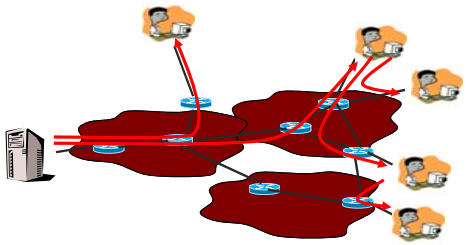
Scaling Problem

- Millions of clients ⇒ server and network meltdown



29

P2P System



- Leverage the resources of client machines (peers)
  - Computation, storage, bandwidth

30

Why p2p?

- Harness lots of spare capacity
  - 1 Big Fast Server: 1Gbit/s, $10k/month++
  - 2,000 cable modems: 1Gbit/s, $??
  - 1M end-hosts: Uh, wow.
  - Capacity grows with the number of users!
- Build very large-scale, self-managing systems
  - Same techniques useful for companies and p2p apps
    - E.g., Akamai's 14,000+ nodes, Google's 100,000+ nodes
  - Many differences to consider
    - Servers versus arbitrary nodes
    - Hard state (backups!) versus soft state (caches)
    - Security, fairness, freeloading, ..

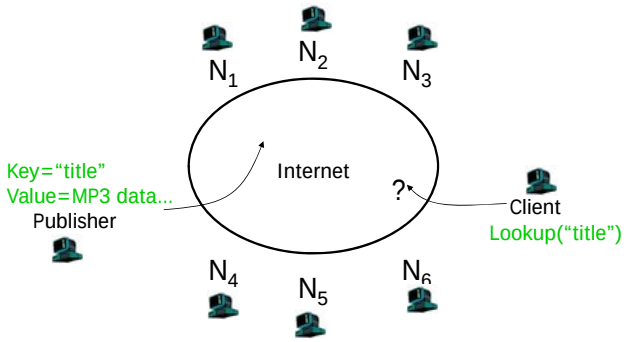
31

Common P2P 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?
- Search tends to be the most challenging:
  - Needles vs. Haystacks
    - Searching for top 40, or an obscure punk track from 1981 that nobody ever heard of?
  - Search expressiveness: Whole word? Regular expressions? File names? Attributes? Whole-text? …
    - E.g., p2p gnutella or p2p google?

32

Searching



33

What is (was) out there?

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

34

The Solution Space

- **Centralized Database**
  - Napster
- Query Flooding
  - Gnutella
- Intelligent Query Flooding
  - KaZaA
- Swarming
  - BitTorrent
- Structured Overlay Routing
  - Distributed Hash Tables

35

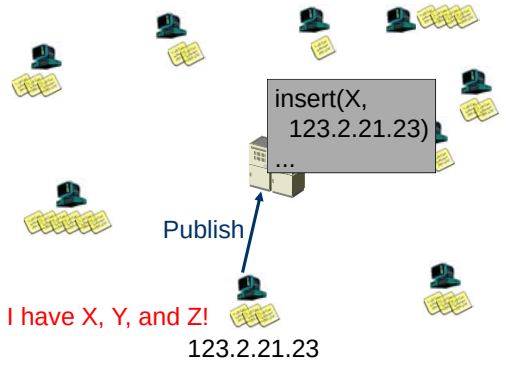
Napster: Centralized Database

- Operational from 1999 to 2001
  - Peaked at 1.5 million simultaneous users

- **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

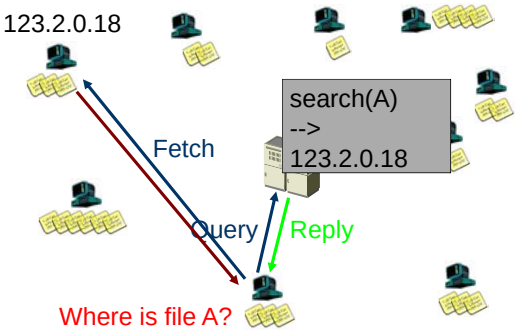
36

Napster: Publish



37

Napster: Search



38

Napster: Discussion

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

39

Next Topic...

- **Centralized Database**
  - Napster
- **Query Flooding**
  - Gnutella
- **Intelligent Query Flooding**
  - KaZaA
- **Swarming**
  - BitTorrent
- **Structured Overlay Routing**
  - Distributed Hash Tables

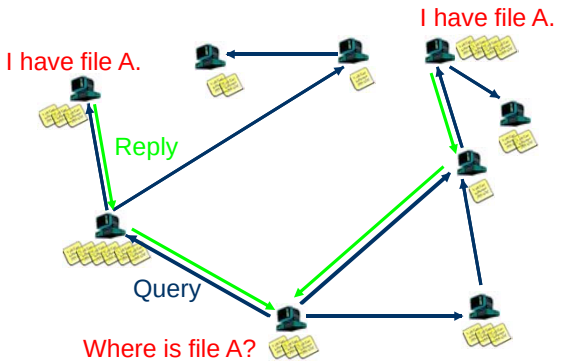
40

Gnutella: Flooding

- Released in 2000 and improved over time; still alive
  - Many clients available, very popular
- **Join**: on startup, client contacts a few other nodes; these become its "neighbors"
- **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

41

Gnutella: Search



42

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

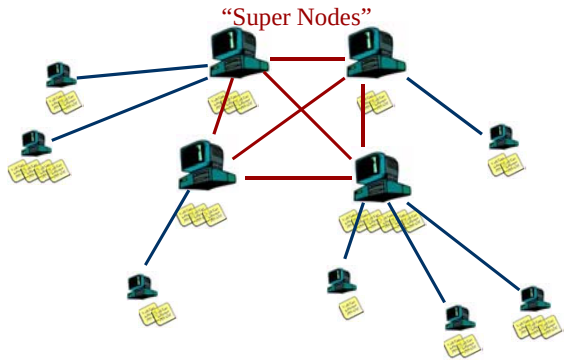
43

KaZaA: Query Flooding

- First released in 2001 and still used today
  - Also very popular
- **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

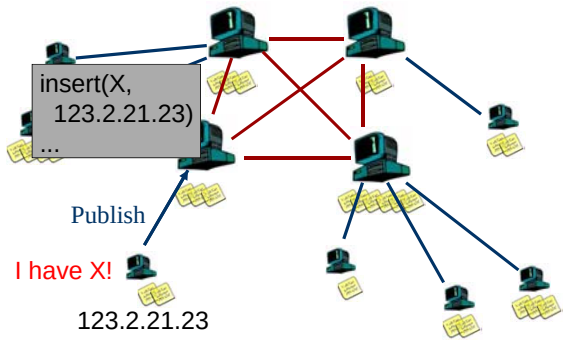
44

KaZaA: Network Design



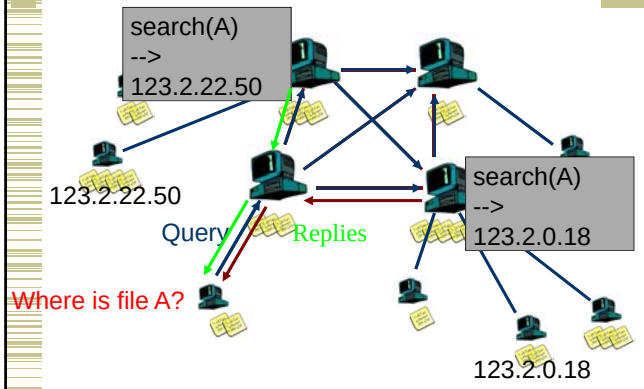
45

KaZaA: File Insert



46

KaZaA: File Search



47

KaZaA: Fetching

- More than one node may have requested 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

48

KaZaA: Discussion

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

Stability and Superpeers

- 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
  - Caching effect
    - Requires network stability
- Superpeer selection is time-based
  - How long you have been on is a good predictor of how long you will be around

The Solution Space

- **Centralized Database**
  - Napster
- **Query Flooding**
  - Gnutella
- **Intelligent Query Flooding**
  - KaZaA
- **Swarming**
  - BitTorrent
- **Structured Overlay Routing**
  - Distributed Hash Tables

- More on Thursday …

Napster: History

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

Gnutella: History

- 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”

67

KaZaA: History

- In 2001, KaZaA created by Dutch company Kazaa BV
- Single network called FastTrack used by other clients as well: Morpheus, giFT, etc.
- Eventually protocol changed so other clients could no longer talk to it
- Very popular file sharing network today with >10 million users (number varies)

68