CS 640 Introduction to Computer Networks

Lecture30

Today's lecture

- Peer to peer applications
  - Napster
  - Gnutella
  - KaZaA
  - Chord

What is P2P?

- Significant autonomy from central servers
- Exploits resources at the edges of Internet
  - Bandwidth
  - Storage
  - Processing
- Resources at edge have intermittent connectivity
- Dynamic joins and leaves

Applications

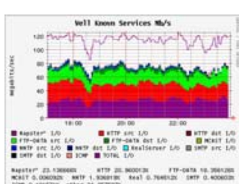
- P2P file sharing
  - Napster, Gnutella, KaZaA, etc.
- Storage and lookup
  - Chord, CAN, etc.
- P2P communication
  - Instant messaging
- P2P computation
  - seti@home

P2P file sharing software

- Allows Alice to open up a directory in her file system
  - Anyone can retrieve a file from directory
  - Like a Web server
- Allows Alice to copy files from other users' open directories:
  - Like a Web client
- Allows users to search the peers for content based on keyword matches:
  - Like Google

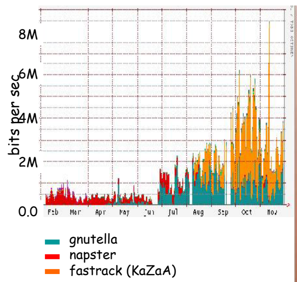
Napster

- The most (in)famous file sharing program
  - 5/99: Shawn Fanning (freshman, Northeasten U.) founds Napster Online music service
  - 12/99: First lawsuit
  - 3/00: 25% UW traffic Napster
  - 2/01: US Circuit Court of Appeals: Napster knew users violating copyright laws
  - 7/01: Simultaneous online: Napster 160K, Gnutella: 40K, Morpheus (KaZaA): 300K



Napster

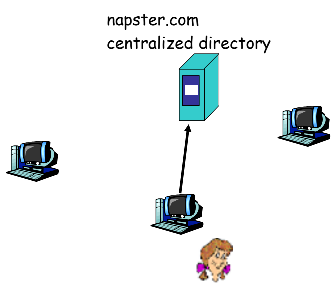
- Judge orders Napster to pull plug in July ‘01
- Other file sharing apps take over!



CS 640

How Napster works

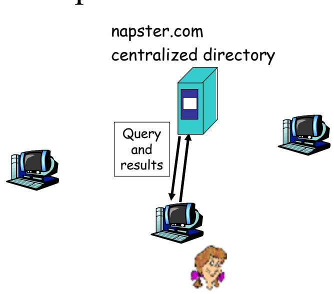
1. File list and IP address is uploaded



CS 640

How Napster works

2. User requests search at server.

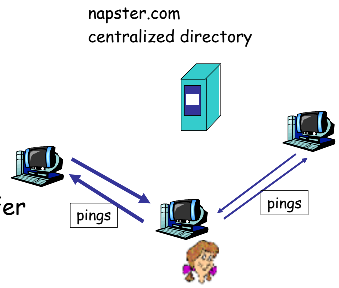


CS 640

How Napster works

3. User pings hosts that apparently have data.

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

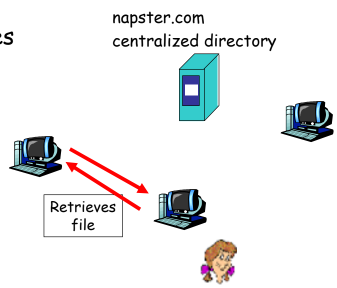


CS 640

How Napster works

4. User chooses server

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



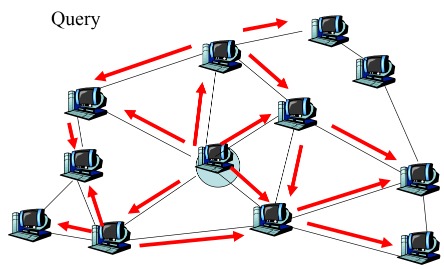
CS 640

2. Unstructured P2P File Sharing

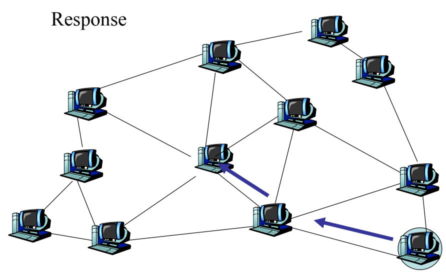
- Napster
- Gnutella
- KaZaA
- Chord

CS 640

Distributed Search/Flooding



Distributed Search/Flooding



Gnutella

- Focus: decentralize search 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

Gnutella

- Gnutella history:
  - 3/14/00: release by AOL, almost immediately withdrawn
  - Became open source
  - Many iterations to fix poor initial design (poor design turned many people off)
- Issues:
  - How much traffic does one query generate?
  - How many hosts can it support at once?
  - What is the latency associated with querying?
  - Is there a bottleneck?

CS 640

Gnutella: limited scope query

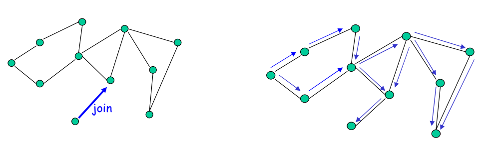
Searching by flooding:

- 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 (not files)

CS 640

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



CS 640

Gnutella in practice

- Gnutella traffic << KaZaA traffic
- KaZaA:
  - hierarchy, queue management, parallel download,…

CS 640

Gnutella Discussion:

- researchers like it because it's open source
  - but is it truly representative?
- architectural lessons learned?
- good source for technical info/open questions:
  http://www.limewire.com/index.jsp/tech_papers

CS 640

2. Unstructured P2P File Sharing

- Napster
- Gnutella
- KaZaA
- Chord

CS 640

KaZaA: The service

- More than 3 million up peers sharing over 3,000 terabytes of content
- More popular than Napster ever was
- More than 50% of Internet traffic?
- MP3s & entire albums, videos, games
- Optional parallel downloading of files
- Automatically switch to new download server when current server becomes unavailable
- Provides estimated download times

KaZaA: The service (2)

- User can configure max number of simultaneous uploads and max number of simultaneous downloads
- Queue management at server and client
  - Frequent uploaders can get priority in server queue
- Keyword search
  - User can configure “up to x” responses to keywords
- Responses to keyword queries come in waves; stops when x responses are found
- To user, service resembles Google, but provides links to MP3s and videos, not Web pages

KaZaA: Technology

Software

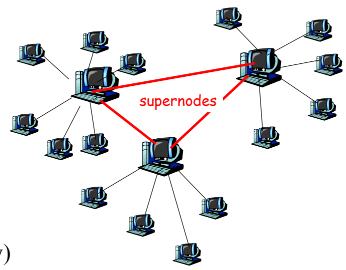
- Proprietary, files and control data encrypted
- Hints:
  - KaZaA Web site gives a few
  - Reverse engineering attempts described on Web
- Everything is HTTP requests and responses

Architecture

- Hierarchical
- Cross between Napster and Gnutella

KaZaA: Architecture

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



CS 640

KaZaA: Architecture (2)

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

CS 640

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

CS 640

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

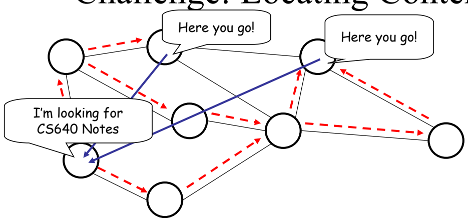
Parallel Downloading; Recovery

- 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

3. Structured P2P: DHT Approaches

- Want a storage and lookup service with better service guarantees and more efficient
- A Distributed Hash Table (DHT)
  - Chord
  - CAN
  - Pastry
  - Tapestry

Challenge: Locating Content



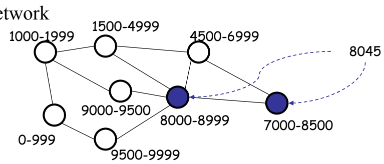
- Simplest strategy: expanding ring search
  - If K of N nodes have copy, expected search cost *at least* N/K, i.e., O(N)
  - Need many cached copies to keep search overhead small

Directed Searches

- Idea:
  - Assign particular nodes to hold particular content (or pointers to it, like an information booth)
  - When a node wants that content, go to the node that is supposed to have or know about it
- Challenges:
  - Distributed: want to distribute responsibilities among existing nodes in the overlay
  - Adaptive: nodes join and leave the P2P overlay
    - distribute knowledge responsibility to joining nodes
    - redistribute knowledge responsibility from those leaving

DHT Step 1: The Hash

- Introduce a hash function to map the object being searched for to a unique identifier:
  - e.g., h("CS640 Class notes") → 8045
- Distribute the range of the hash function among all nodes in the network



- Each node must "know about" at least one copy of each object that hashes within its range (when one exists)

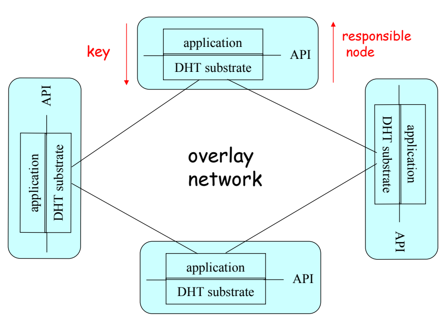
DHT Step 2: Routing

- For each object, node(s) whose range(s) cover that object must be reachable via a "short" path by any querying node
- Different approaches for routing requests
  - (CAN,Chord,Pastry,Tapestry) differ fundamentally only in the routing approach
  - They all rely on hash functions to map objects

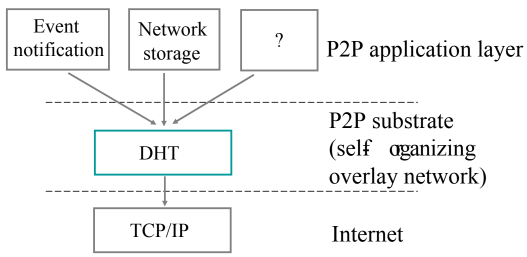
DHT API

- each data item (e.g., file or metadata with pointers) has a key in some ID space
- In each node, DHT software provides API:
  - Application gives API key k
  - API returns IP address of node responsible for k
- API is implemented with an underlying DHT overlay and distributed algorithms

DHT API



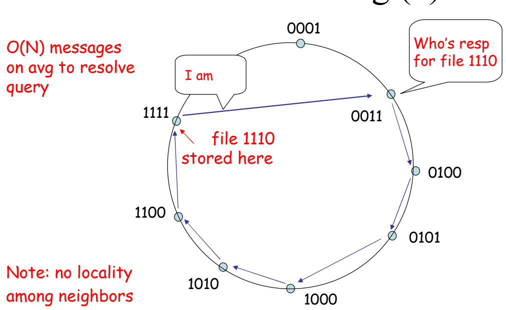
DHT Layered Architecture



Consistent hashing (1)

- Overlay network is a circle
- Each node has randomly chosen id
  - Keys in same id space
- Node's successor in circle is node with next largest id
  - Each node knows IP address of its successor
- Key is stored in closest successor

Consistent hashing (2)



Consistent hashing (3)

Node departures

- Each node must track s ≥ 2 successors
- If your successor leaves, take next one
- Ask your new successor for list of its successors; update your s successors

Node joins

- You're new, node id k
- Ask any node n to find the node n' that is the successor for id k
- Get successor list from n'
- Tell your predecessors to update their successor lists
- Thus, each node must track its predecessor

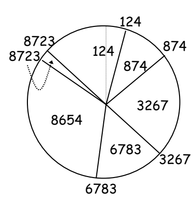
Consistent hashing (4)

- Overlay is actually a circle with small chords for tracking predecessor and k successors
- # of neighbors = s+1: O(1)
  - The ids of your neighbors along with their IP addresses is your "routing table"
- Average # of messages to find key is O(N)

Can we do better?

Chord

- Nodes assigned 1-dimensional IDs in hash space at random (e.g., hash on IP address)
- Consistent hashing: Range covered by node is from previous ID up to its own ID (modulo the ID space)

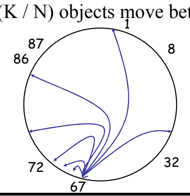


Chord Routing

- A node s's i^{th} neighbor has the ID that is equal to $s+2^i$ or is the next largest ID (mod ID space), $i \geq 0$
- To reach the node handling ID t, send the message to neighbor $\#\log_2(t\text{-}s)$
- Requirement: each node s must know about the next node that exists clockwise on the Chord (0^{th} neighbor)
- Set of known neighbors called a finger table

Chord Routing (cont'd)

- A node s is node t's neighbor if s is the closest node to $t+2^i$ mod H for some i. Thus,
  - each node has at most $\log_2 N$ neighbors
  - for any object, the node whose range contains the object is reachable from any node in no more than $\log_2 N$ overlay hops
- Given K objects, with high probability each node has at most $(1 + \log_2 N)$ K / N in its range
- When a new node joins or leaves the overlay, O(K / N) objects move between nodes



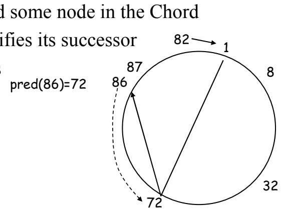
i	Finger table for node 67
0	72
1	72
2	72
3	86
4	86
5	1
6	32

Closest node clockwise to $67+2^i$ mod 100

Chord Node Insertion

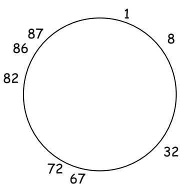
- One protocol addition: each node knows its closest counter-clockwise neighbor
- A node selects its unique (pseudo-random) ID and uses a bootstrapping process to find some node in the Chord
- Using Chord, the node identifies its successor
- A new node's predecessor is its successor's former predecessor

Example: Insert 82



Chord Node Insertion (cont'd)

- First: set added node s's fingers correctly
  - s's predecessor t does the lookup for each distance of 2^i from s

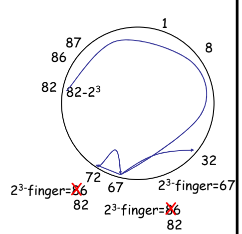


Lookups from node 72
Lookup(83) = 86
Lookup(84) = 86
Lookup(86) = 86
Lookup(90) = 1
Lookup(98) = 1
Lookup(14) = 32
Lookup(46) = 67

i	Finger table for node 82
0	86
1	86
2	86
3	1
4	1
5	32
6	67

Chord Node Insertion (cont'd)

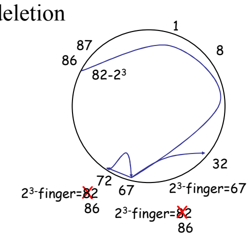
- Next, update other nodes' fingers about the entrance of s (when relevant). For each i:
  - Locate the closest node to s (counter-clockwise) whose 2^i-finger can point to s: largest possible is s - 2^i
  - Use Chord to go (clockwise) to largest node t before or at s - 2^i
    - route to s - 2^i, if arrived at a larger node, select its predecessor as t
  - If t's 2^i-finger routes to a node larger than s
    - change t's 2^i-finger to s
    - set t = predecessor of t and repeat
  - Else i++, repeat from top
- O($\log^2$ N)



e.g., for i=3

Chord Node Deletion

- Similar process can perform deletion



e.g., for i=3