Peer to Peer Systems

Arvind Krishnamurthy

Fall 2003

Peer to Peer Systems

- Typically each member stores/provides access to content
- Has quickly grown in popularity
- Goals:
  - Fast search and retrieval
  - Scalability
    - Avoid centralization
    - Limit amount of state on each node
    - Limit number of messages in the system
- Traditional networking systems with similar goals:
  - Routing, DNS

Outline

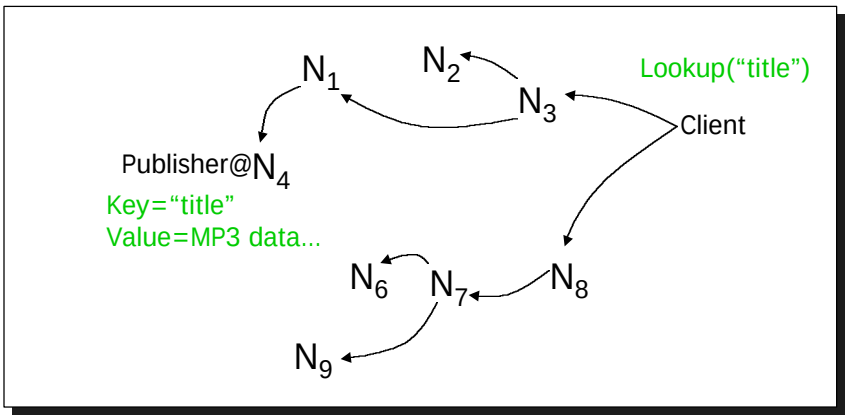
- Unstructured systems:
  - Centralized Napster-like design
  - Decentralized Gnutella-like design
  - Decentralized system with routing tables (Freenet)
- Structured systems:
  - Distributed hash tables: Chord, CAN, Tapestry
  - Other distributed data structures: Skip Graphs

Centralized Napster

- Simple centralized scheme → motivated by ability to control
- How to find a file:
  - On startup, client contacts central server and reports list of files
  - Query the index system → return a machine that stores the required file
    - Ideally this is the closest/least-loaded machine
  - Fetch the file directly from peer
- Advantages:
  - Simple
  - Easy to implement sophisticated search engines on top of the index system
- Disadvantages:
  - Robustness, scalability
  - Easy to sue!

Decentralized system: Gnutella

- How to find a file: send request to all neighbors
  - Neighbors recursively forward the requests
  - Eventually find a node that has the file



Decentralized flooding

- Advantages:
  - Totally decentralized, highly robust
- Disadvantages:
  - Not scalable; the entire network can be swamped with request (to alleviate this problem, each request has a TTL)
  - Especially hard on slow clients
    - At some point broadcast traffic on Gnutella exceeded 56kbps – what happened?
    - Modem users were effectively cut off!

Decentralized flooding: Gnutella

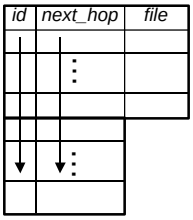
- Basic message header
  - Unique ID, TTL, Hops
- Message types
  - On startup, client contacts any servent (**serv**er + cli**ent**) in network
    - Servent interconnection used to forward control (queries, hits, etc)
  - Ping – probes network for other servents
  - Pong – response to ping, contains IP addr, # of files, etc.
  - Query – search criteria + speed requirement of servent
  - QueryHit – successful response to Query, contains addr + port to transfer from, speed of servent, etc.
- Ping, Queries are flooded
- QueryHit, Pong: reverse path of previous message

Routing Based Lookups

- Example system: Freenet
- Files are stored according to associated key
  - Core idea: try to cluster information about similar keys
- Messages
  - Random 64bit ID used for loop detection
    - Each node maintains the list of query IDs that have traversed it → help to avoid looping messages
  - TTL
    - TTL is decremented each time the query message is forwarded
- Addition goals to file location:
  - Provide publisher anonymity, security

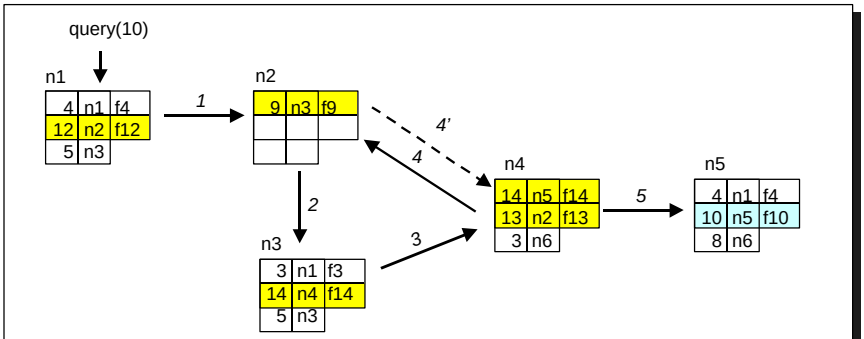
Routing Tables

- ID: file identifier
- Next_hop: another node that stores the file id



- Forwarding of query for file id:
  - If file id is stored locally, then stop
    - Forward data back to upstream requestor
    - Requestor adds file to cache and entry into routing table
  - If not, search for the "closest" id in the table and forward request
  - If data is not found, report failure
    - Requestor then tries next closest match in routing table

Routing: Freenet Example



Note: doesn't show file caching on the reverse path

Freenet discussion

- Is this approach scalable?
  - What happens after the system runs for a while?
  - How do routing tables change?