15-744: Computer Networking

L-22: P2P

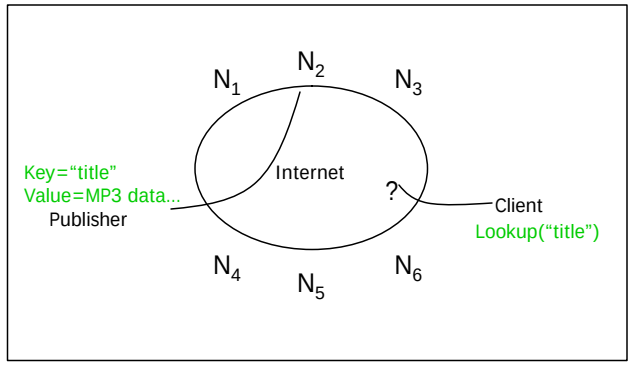
Peer-to-Peer Networks

- Typically each member stores/provides access to content
- Has quickly grown in popularity
  - Bulk of traffic from/to CMU is Kazaa!
- Basically a replication system for files
  - Always a tradeoff between possible location of files and searching difficulty
  - Peer-to-peer allow files to be anywhere → searching is the challenge
  - Dynamic member list makes it more difficult
- What other systems have similar goals?
  - Routing, DNS

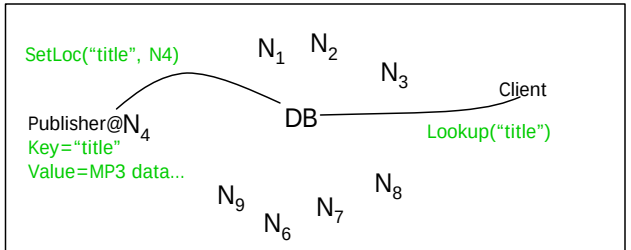
Overview

- P2P Lookup Overview
- Centralized/Flooded Lookups
- Routing-based Lookups
- CMU Research in P2P

The Lookup Problem

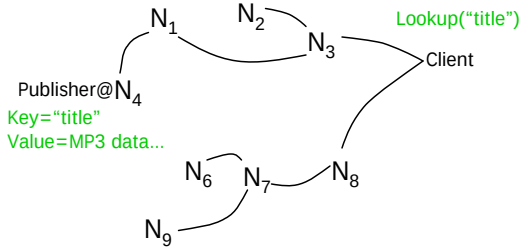


Centralized Lookup (Napster)



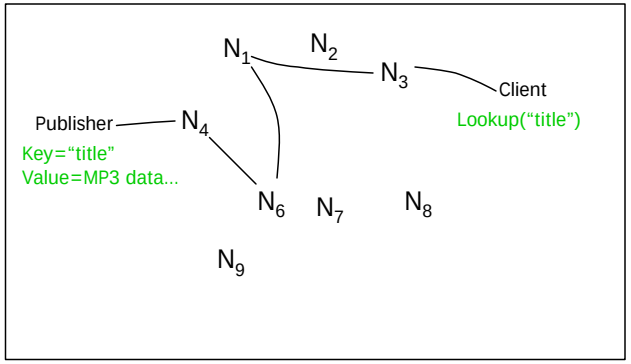
Simple, but O(N) state and a single point of failure

Flooded Queries (Gnutella)



Robust, but worst case O(N) messages per lookup

Routed Queries (Freenet, Chord, etc.)



Overview

- P2P Lookup Overview
- Centralized/Flooded Lookups
- Routing-based Lookups
- CMU Research in P2P

Centralized: Napster

- Simple centralized scheme → motivated by ability to sell/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

Centralized: Napster

- Advantages:
  - Simple
  - Easy to implement sophisticated search engines on top of the index system
- Disadvantages:
  - Robustness, scalability
  - Easy to sue!

Flooding: Gnutella

- On startup, client contacts any servent (**serv**er + cli**ent**) in network
  - Servent interconnection used to forward control (queries, hits, etc)
- Idea: broadcast the request
- How to find a file:
  - Send request to all neighbors
  - Neighbors recursively forward the request
  - Eventually a machine that has the file receives the request, and it sends back the answer
  - Transfers are done with HTTP between peers

Flooding: Gnutella

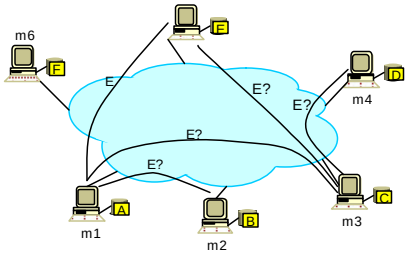
- Advantages:
  - Totally decentralized, highly robust
- Disadvantages:
  - Not scalable; the entire network can be swamped with request (to alleviate this problem, each request has a TTL)
  - Especially hard on slow clients
    - At some point broadcast traffic on Gnutella exceeded 56kbps – what happened?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

Flooding: Gnutella Details

- Basic message header
  - Unique ID, TTL, Hops
- Message types
  - Ping – probes network for other servents
  - Pong – response to ping, contains IP addr, # of files, # of Kbytes shared
  - Query – search criteria + speed requirement of servent
  - QueryHit – successful response to Query, contains addr + port to transfer from, speed of servent, number of hits, hit results, servent ID
  - Push – request to servent ID to initiate connection, used to traverse firewalls
- Ping, Queries are flooded
- QueryHit, Pong, Push reverse path of previous message

Flooding: Gnutella Example

Assume: m1's neighbors are m2 and m3; m3's neighbors are m4 and m5;…



Flooding: FastTrack (aka Kazaa)

- Modifies the Gnutella protocol into two-level hierarchy
- Supernodes
  - Nodes that have better connection to Internet
  - Act as temporary indexing servers for other nodes
  - Help improve the stability of the network
- Standard nodes
  - Connect to supernodes and report list of files
  - Allows slower nodes to participate
- Search
  - Broadcast (Gnutella-style) search across supernodes
- Disadvantages
  - Kept a centralized registration → allowed for law suits ☹

Overview

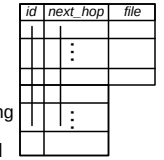
- P2P Lookup Overview
- Centralized/Flooded Lookups
- Routing-based Lookups
- CMU Research in P2P

Routing: Freenet

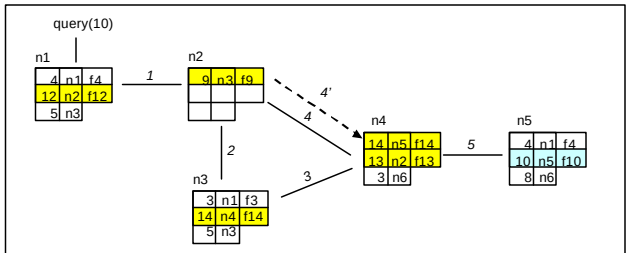
- Addition goals to file location:
  - Provide publisher anonymity, security
- Files are stored according to associated key
  - Core idea: try to cluster information about similar keys
- Messages
  - Random 64bit ID used for loop detection
    - Each node maintains the list of query IDs that have traversed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

Routing: Freenet Routing Tables

- *id* – file identifier
- *next_hop* – another node that stores the file id
- *file* – file identified by *id* being stored on the local node
- Forwarding of query for file *id*
  - If file *id* stored locally, then stop
    - Forward data back to upstream requestor
    - Requestor adds file to cache, adds entry in routing table
  - If not, search for the "closest" *id* in the stack, and forward the message to the corresponding *next_hop*
  - If data is not found, failure is reported back
    - Requestor then tries next closest match in routing table



Routing: Freenet Example



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

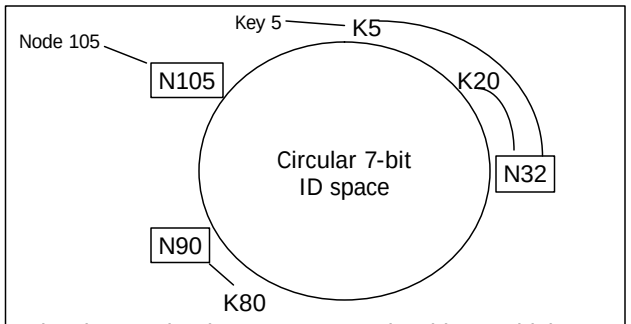
Routing: Structured Approaches

- Goal: make sure that an item (file) identified is always found in a reasonable # of steps
- Abstraction: a distributed hash-table (DHT) data structure
  - insert(id, item);
  - item = query(id);
  - Note: item can be anything: a data object, document, file, pointer to a file…
- Proposals
  - CAN (ICIR/Berkeley)
  - Chord (MIT/Berkeley)
  - Pastry (Rice)
  - Tapestry (Berkeley)

Routing: Chord

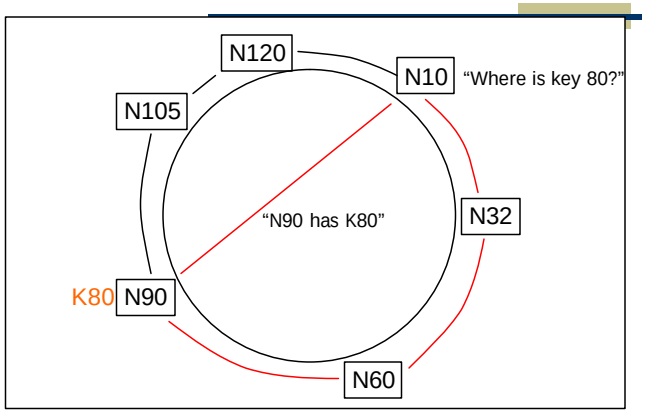
- Associate to each node and item a unique *id* in an *uni*-dimensional space
- Properties
  - Routing table size O(log(*N*)) , where *N* is the total number of nodes
  - Guarantees that a file is found in O(log(*N*)) steps

Aside: Consistent Hashing [Karger 97]

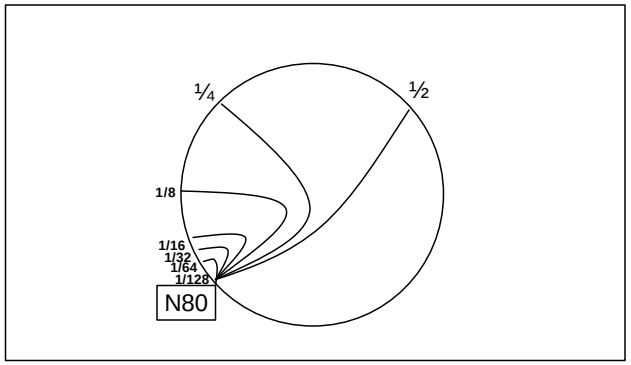


A key is stored at its successor: node with next higher ID

Routing: Chord Basic Lookup



Routing: "Finger table" - Faster Lookups

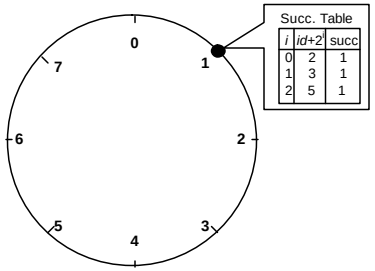


Routing: Chord Summary

- Assume identifier space is $0 \ldots 2^m$
- Each node maintains
  - Finger table
    - Entry *i* in the finger table of *n* is the first node that succeeds or equals $n + 2^i$
  - Predecessor node
- An item identified by *id* is stored on the successor node of *id*

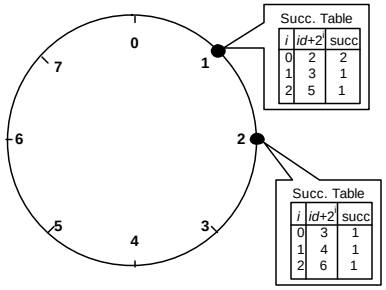
Routing: Chord Example

- Assume an identifier space 0..8
- Node n1:(1) joins→all entries in its finger table are initialized to itself



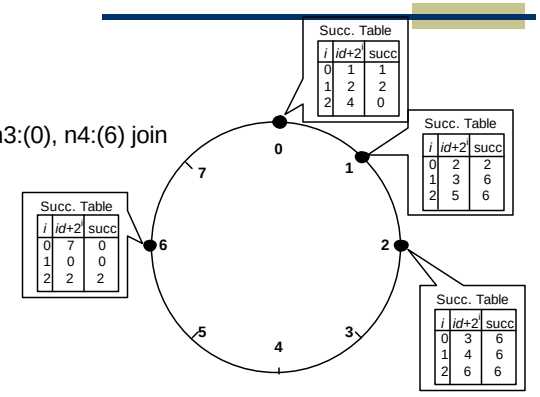
Routing: Chord Example

- Node n2:(3) joins



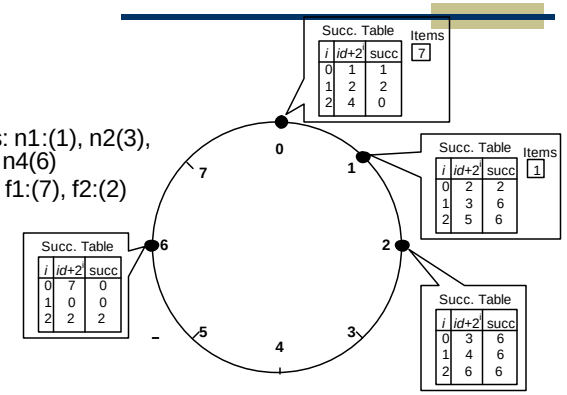
Routing: Chord Example

- Nodes n3:(0), n4:(6) join



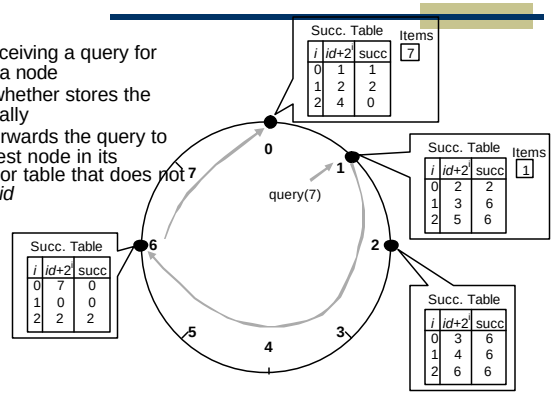
Routing: Chord Examples

- Nodes: n1:(1), n2(3), n3(0), n4(6)
- Items: f1:(7), f2:(2)



Routing: Query

- Upon receiving a query for item *id*, a node
- Check whether stores the item locally
- If not, forwards the query to the largest node in its successor table that does not exceed *id*



Performance Concerns

- Each hop in a routing-based P2P network can be expensive
  - No correlation between neighbors and their location
  - A query can repeatedly jump from Europe to North America, though both the initiator and the node that store the item are in Europe!

Summary

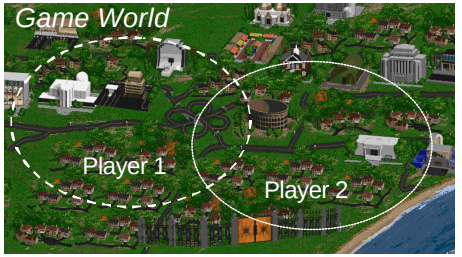
- The key challenge of building wide area P2P systems is a scalable and robust location service
- Solutions covered in this lecture
  - Naptser: centralized location service
  - Gnutella: broadcast-based decentralized location service
  - Freenet: intelligent-routing decentralized solution (but correctness not guaranteed; queries for existing items may fail)
  - CAN, Chord, Tapestry, Pastry: intelligent-routing decentralized solution
    - Guarantee correctness
    - Tapestry (Pastry ?) provide efficient routing, but more complex

Overview

- P2P Lookup Overview
- Centralized/Flooded Lookups
- Routing-based Lookups
- CMU Research in P2P

What Do Games Look Like?

- Large shared world
  - Composed of map information, textures, etc
  - Populated by active entities: user avatars, computer AI's, etc
- Only parts of world relevant to particular user/player



Individual Player's View

- Interactive environment (e.g. door, rooms)
- Live ammo
- Monsters
- Players
- Game state

Current Game Architectures

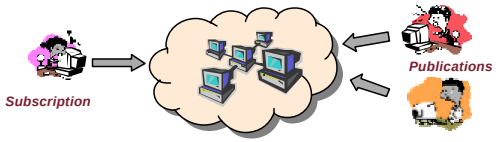
- Centralized client-server (e.g., Quake)
  - Every update sent to server who maintains "true" state
- Advantages/disadvantages
  - \+ Reduces client bandwidth requirements
  - \+ State management, cheat proofing much easier
  - \- Server bottleneck for computation and bandwidth → current games limited to about 6000 players
  - \- Single point of failure
  - \- Response time limited by client-server latency

Doesn't scale well

Goal: A P2P Multiplayer Game

- Allow 1000's of people to interact in a single virtual world
- Key requirements
  - Robustness: node failures
  - Scalability: number of participants & size of virtual world
  - Performance: interaction quality should only be limited by capabilities of nodes and connectivity between them
  - Extensible: should be possible to add to virtual world

What is Publish-Subscribe ?



- Publishers produce events or *publications*
- Subscribers register their interests via *subscriptions*
- Network performs routing such that
  - Publications "meet" subscriptions
  - Publications delivered to appropriate subscribers

Mercury

- A P2P publish-subscribe system
- Query language
  - Type, attribute name, operator, value
    - Example: int x = 200
  - Attribute-values are sortable
- Sufficient for modeling games
  - Game arenas
  - Player statistics, etc

Modeling a Game