15-744: Computer Networking

L-22: P2P

P2P

- Peer-to-peer networks
- Assigned reading
  - [Cla00] Freenet: A Distributed Anonymous Information Storage and Retrieval System
  - [S+01] Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications

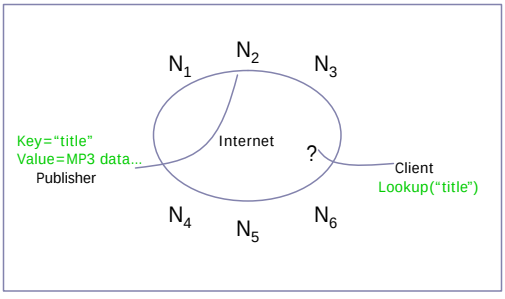
Overview

- P2P Lookup Overview
- Routed/Flooded Lookups
- Distributed Hash Table Lookups
  - Chord
  - CAN

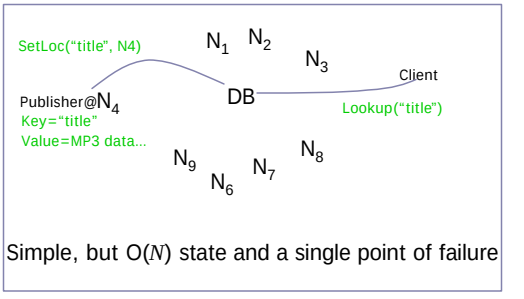
Peer-to-Peer Networks

- Typically each member stores content that it desires
- 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

The Lookup Problem

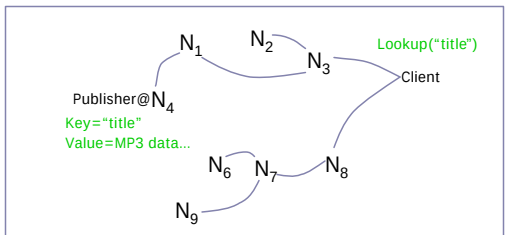


Centralized Lookup (Napster)



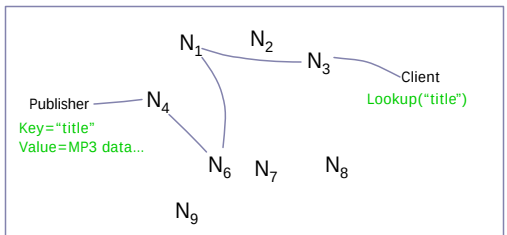
Simple, but O(N) state and a single point of failure

Flooded Queries (Gnutella)



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

Routed Queries (Freenet, Chord, etc.)



Overview

- P2P Lookup Overview
- Routed/Flooded Lookups
- Distributed Hash Table Lookups
  - Chord
  - CAN

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
- Advantages:
  - Simplicity, easy to implement sophisticated search engines on top of the index system
- Disadvantages:
  - Robustness, scalability

Gnutella

- Distribute file location
- Idea: multicast the request
- Hot to find a file:
  - Send request to all neighbors
  - Neighbors recursively multicast the request
  - Eventually a machine that has the file receives the request, and it sends back the answer
- 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)

Gnutella

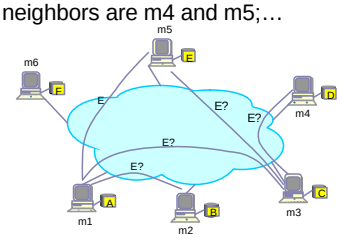
- On startup client contacts any servent (server + client) in network
  - Servent interconnection used to forward control (queries, hits, etc)
- Idea: multicast the request
- How to find a file:
  - Send request to all neighbors
  - Neighbors recursively multicast the request
  - Eventually a machine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

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

Gnutella: Example

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



Freenet

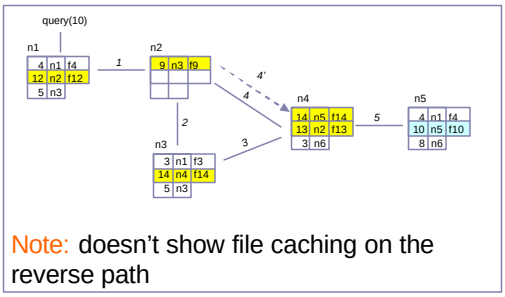
- Addition goals to file location:
  - Provide publisher anonymity, security
  - Resistant to attacks – a third party shouldn't be able to deny the access to a particular file (data item, object), even if it compromises a large fraction of machines
- Files are stored according to associated key
  - Core idea: try to cluster information about similar keys
- Messages
  - Random 64bit ID used for loop detection
  - TTL
    - TTL 1 are forwarded with finite probablity
    - Helps anonymity
  - Depth counter
    - Opposite of TTL – incremented with each hop
    - Depth counter initialized to small random value

Data Structure

- Each node maintains a common stack
  - *id* – file identifier
  - *next_hop* – another node that store the file id
  - *file* – file identified by *id* being stored on the local node
- Forwarding:
  - Each message contains the file *id* it is referring to
  - If file *id* stored locally, then stop
    - Forwards 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*

id	*next_hop*	*file*
	⋮	
	⋮	

Query Example

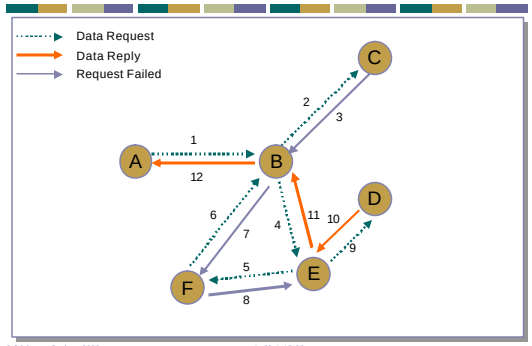


Note: doesn't show file caching on the reverse path

Freenet Requests

- Any node forwarding reply may change the source of the reply (to itself or any other node)
  - Helps anonymity
- Each query is associated a TTL that is decremented each time the query message is forwarded; to obscure distance to originator:
  - TTL can be initiated to a random value within some bounds
  - When TTL=1, the query is forwarded with a finite probability
- Each node maintains the state for all outstanding queries that have traversed it → help to avoid cycles
- If data is not found, failure is reported back
  - Requestor then tries next closest match in routing table

Freenet Request



Freenet Search Features

- Nodes tend to specialize in searching for similar keys over time
  - Gets queries from other nodes for similar keys
- Nodes store similar keys over time
  - Caching of files as a result of successful queries
- Similarity of keys does not reflect similarity of files
- Routing does not reflect network topology

Freenet File Creation

- Key for file generated and searched → helps identify collision
  - Not found ("All clear") result indicates success
  - Source of insert message can be change by any forwarding node
- Creation mechanism adds files/info to locations with similar keys
- New nodes are discovered through file creation
- Erroneous/malicious inserts propagate original file further

Cache Management

- LRU Cache of files
- Files are not guaranteed to live forever
  - Files "fade away" as fewer requests are made for them
- File contents can be encrypted with original text names as key
  - Cache owners do not know either original name or contents → cannot be held responsible

Freenet Naming

- Freenet deals with keys
  - But humans need names
  - Keys are flat → would like structure as well
- Could have files that store keys for other files
  - File /text/philiosophy could store keys for files in that directory → how to update this file though?
- Search engine → undesirable centralized solution

Freenet Naming - Indirect files

- Normal files stored using content-hash key
  - Prevents tampering, enables versioning, etc.
- Indirect files stored using name-based key
  - Indirect files store keys for normal files
  - Inserted at same time as normal file
- Has same update problems as directory files
  - Updates handled by signing indirect file with public/private key
  - Collisions for insert of new indirect file handled specially → check to ensure same key used for signing
- Allows for files to be split into multiple smaller parts

Overview

- P2P Lookup Overview
- Routed/Flooded Lookups
- Distributed Hash Table Lookups
  - Chord
  - CAN

Other Solutions to Location Problem

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

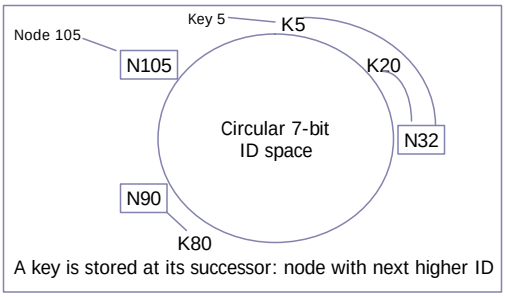
Chord

- 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

Data Structure

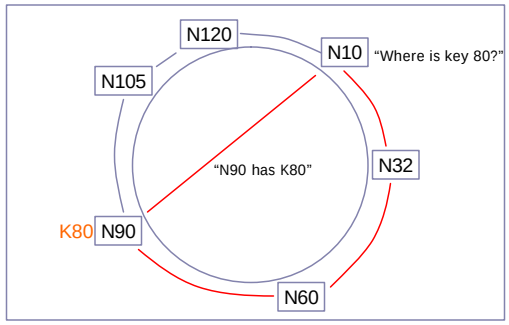
- Assume identifier space is $0…2^m$
- Each node maintains
  - Finger table
    - Entry *i* in the finger table of *n* is the first node that succeeds or equals $n + 2^i$
  - Predecessor node
- An item identified by *id* is stored on the succesor node of *id*

Consistent Hashing [Karger 97]



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

Basic Lookup

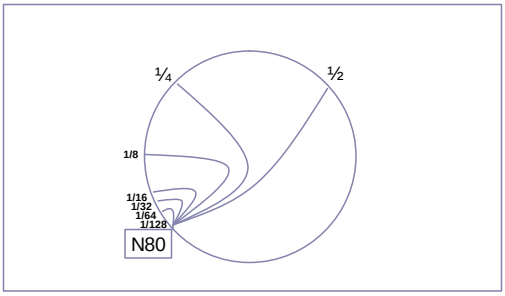


Simple Lookup Algorithm

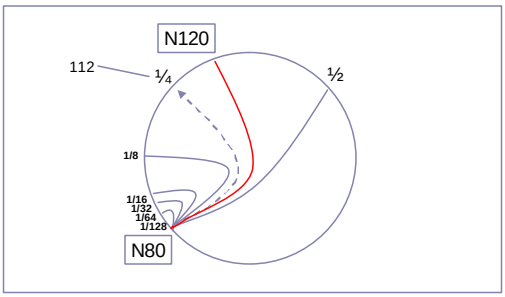
```
Lookup(my-id, key-id)
  n = my successor
  if my-id < n < key-id
      call Lookup(id) on node n   // next hop
  else
      return my successor         // done
```

- Correctness depends only on successors

"Finger table" - log(N)-time lookups



Finger i Points to Successor of n+2i

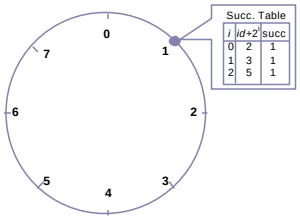


Lookup with Fingers

```
Lookup(my-id, key-id)
  look in local finger table for
      highest node n s.t. my-id < n < key-id
  if n exists
      call Lookup(id) on node n     // next hop
  else
      return my successor           // done
```

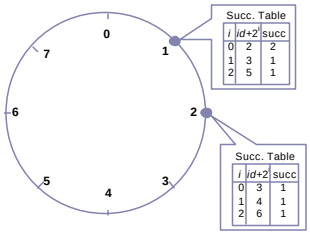

Chord Example

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



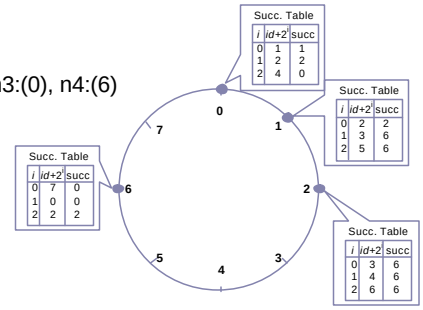
Chord Example

- Node n2:(3) joins



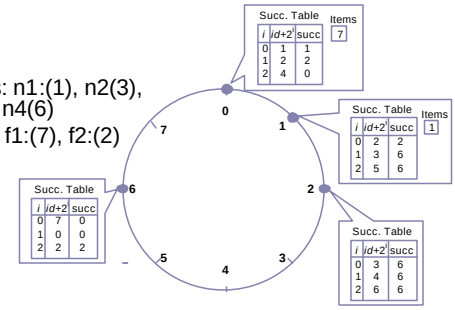
Chord Example

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



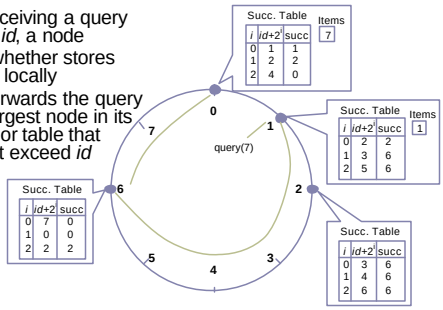
Chord Examples

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



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*



Overview

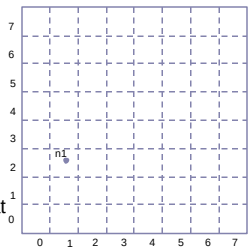
- P2P Lookup Overview
- Routed/Flooded Lookups
- Distributed Hash Table Lookups
  - Chord
  - CAN

CAN

- Associate to each node and item a unique *id* in an *d*-dimensional space
  - Virtual Cartesian coordinate space
- Entire space is partitioned amongst all the nodes
  - Every node "owns" a zone in the overall space
- Abstraction
  - Can store data at "points" in the space
  - Can route from one "point" to another
- Point = node that owns the enclosing zone
- Properties
  - Routing table size O(*d*)
  - Guarantees that a file is found in at most $d*n^{1/d}$ steps, where *n* is the total number of nodes

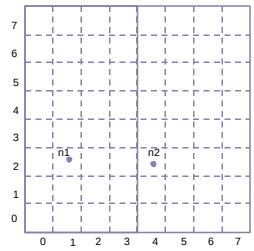
CAN E.g.: Two Dimensional Space

- Space divided between nodes
- All nodes cover the entire space
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1
- Example:
  - Assume space size (8 x 8)
  - Node n1:(1, 2) first node that joins → cover the entire space



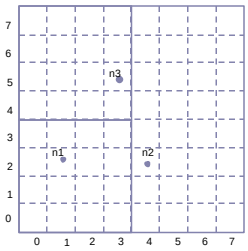
CAN E.g.: Two Dimensional Space

- Node n2:(4, 2) joins → space is divided between n1 and n2



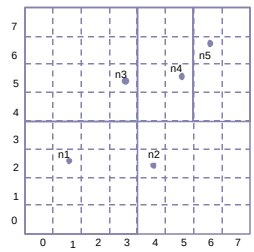
CAN E.g.: Two Dimensional Space

- Node n2:(4, 2) joins → space is divided between n1 and n2



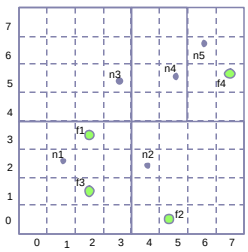
CAN E.g.: Two Dimensional Space

- Nodes n4:(5, 5) and n5:(6,6) join



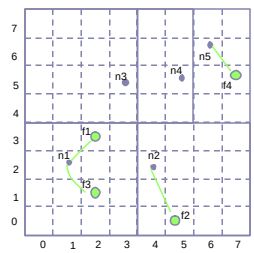
CAN E.g.: Two Dimensional Space

- Nodes: n1:(1, 2); n2:(4,2); n3:(3, 5); n4:(5,5);n5:(6,6)
- Items: f1:(2,3); f2:(5,1); f3:(2,1); f4:(7,5);



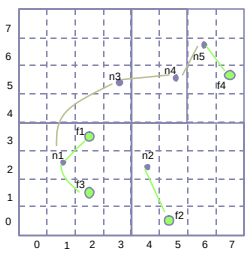
CAN E.g.: Two Dimensional Space

- Each item is stored by the node who owns its mapping in the space



CAN: Query Example

- Each node knows its neighbors in the *d*-space
- Forward query to the neighbor that is closest to the query *id*
- Example: assume n1 queries f4



DHT Concerns

- Performance: routing in the overlay network can be more expensive than in the underlying network
  - Because usually there is no correlation between node ids and their locality; a query can repeatedly jump from Europe to North America, though both the initiator and the node that store the item are in Europe!
  - Solutions: Tapestry takes care of this implicitly; CAN and Chord maintain multiple copies for each entry in their routing tables and choose the closest in terms of network distance

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

Next Lecture: Security

- Denial of service
- IPSec
- Firewalls
- Assigned reading
  - [SWKA00] Practical Network Support for IP Traceback
  - [B89] Security Problems in the TCP/IP Protocol Suite

Important Deadlines

- 4/29 – project writeups due
  - Maximum: 10 pages double column/15 pages single column, reasonable font size, etc.
- 4/24 – final exam (not cumulative)
- 4/29,4/31 – project presentations (15 minutes each)