CSE 123b
Communications Software
Spring 2004
Lecture 13: Peer-to-Peer Networks
Stefan Savage

Some slides courtesy Ion Stoica and Srini Seshan
Quick announcements

- Homework assignment up on the Web site tonight
- New (short) project assigned on Tuesday
Peer-to-peer systems

- Examples
  - Napster, Gnutella, Freenet, KaZaA, CFS, etc.

- Definition?
  - No distinction between client and server
  - All nodes are potential users of a service AND potential providers of a service
Classifications

- What resource is shared?
  - CPU: SETI@Home
  - Storage & BW: most of the rest

- How are resources located?
  - Centralized systems
    » Napster, Seti@Home
  - Distributed systems
    » Unstructured: e.g. Gnutella, KaZaa, eDonkey
    » Structured/routed: e.g. CFS/Chord, Freenet, Overnet

- Search vs Lookup
Challenges

- Dynamic availability—machines go up & down
- Scale—millions of nodes
- Heterogeneity—some have big CPU/mem
- Management—each user controls own node
- Security—what about compromised nodes?
- Fairness—what about freeloaders?
- Performance
The Lookup Problem

Key="title"
Value=MP3 data...
Publisher

Client
Lookup("title")
Centralized Lookup (Napster)

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

- Simple centralized scheme
- How to find a file:
  - On startup, client contacts central server and reports list of files
  - Query the index system
    » Returns addresses of one or more machines that claim to have file
  - Fetch the file directly from peer
- Advantages:
  - Simplicity, easy to implement sophisticated search engines on top of the index system
- Potential disadvantages:
  - Scalability, single point of failure (technical and legal)
Flooded Queries (Gnutella)

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

Key="title"
Value=MP3 data...
Gnutella

- **Distributed** location information
- Self-organizing
  - New node joins network by opening TCP sessions to “neighbors” willing to accept connections
- Idea: *multicast* 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
- Advantages:
  - Totally decentralized, highly robust
- Disadvantages:
  - Not scalable as is; the entire network can be swamped with request (can alleviate some of this problem if each request has a TTL)
Gnutella Details

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

Publisher

Key=“title”
Value=MP3 data...

Client

Lookup(“title”)
Example: Freenet

- **Architecture:**
  - Each file is identified by a *unique* identifier
  - Each machine stores a set of files, and maintains a “routing table” to route the individual requests
  - Use “nearness” metric to get closer to content

- **Additional 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
Freenet Query

- User requests key XYZ – not in local cache
- Looks up nearest key in routing table and forwards to corresponding node
- If request reaches node with data, it forwards data back to upstream requestor
  - Requestor adds file to cache, adds entry in routing table
  - Any node forwarding reply may change the source of the reply → helps anonymity
- If data not found, failure is reported back
Data Structure

- Each node maintains a common stack
  - *id* – file identifier
  - *next_hop* – another node that stores 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
    - 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*
Query Example

Note: doesn’t show file caching on the reverse path
Freenet Summary

- Advantages
  - Totally decentralized architecture → robust and scalable

- Disadvantages
  - Does not always guarantee that a file is found, even if the file is in the network
Example: Chord

- Structured p2p systems: guaranteed lookup
- Associate to each node and item a unique id in a uni-dimensional space
- Goals
  - Scales to hundreds of thousands of nodes
  - Handles rapid arrival and failure of nodes
- 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
Consistent Hashing [Karger 97]

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

IP="198.10.10.1"

Circular 7-bit ID space

K5

K20

N32

K101

K60

N90

N123

Key="LetItBe"
Consistent Hashing costs

- Every node knows of every other node
  - requires global information
- Routing tables are large $O(N)$
- Lookups are fast $O(1)$

Where is “LetItBe”?  
$\text{Hash(“LetItBe”) = K60}$
Chord: basic lookup

Where is “LetItBe”?
Hash(“LetItBe”) = K60

"N90 has K60"

Requires O(N) time
Improvement: “finger tables”

- Every node knows $m$ other nodes in the ring
- Increase distance exponentially
How finger tables work

- Finger $i$ points to successor of $n + 2^i$
Chord lookup

- Lookups take $O(\log N)$ hops
Chord Summary

- $O(\log N)$ guaranteed lookup performance
  - Possible to do even better: $O(\log \log n / \log n)$
- Lookup semantics, **not search**
- **Issues w/simple approach**
  - Reliability & churn (not covered here)
  - Performance: routing over the p2p 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!
    - Partial solution: Weight neighbor nodes by RTT
      - when routing, choose neighbor who is closer to destination with lowest RTT from me
Open p2p issues

- Freeloding problem
  - Does everyone participate?
- Trust?
- Availability/reliability?

- What can we do with p2p that is useful and legal?
Summary

- A key challenge of building wide area P2P systems is a scalable and robust location service
- Solutions covered in this lecture
  - Napster: centralized location service
  - Gnutella: broadcast-based decentralized location service
  - Freenet: intelligent-routing decentralized solution (but correctness not guaranteed; queries for existing items may fail)
  - Chord (and others): intelligent-routing decentralized solution
    - Guarantee correctness
    - Guaranteed worst-case # of operations
    - More fragile
- Lots of open questions