P2P Search Algorithms

吳俊興
國立高雄大學 資訊工程學系
Spring 2006

參考：「同儕計算網路及其應用」課程
Classification of P2P

- **Centralized Systems**
  - Central server to assist administration
    - Napster, Seti@Home

- **Pure Distributed Systems**
  - Unstructured:
    - Gnutella, Freenet
  - Structured
    - CFS/Chord, Tapestry, Pastry

- **Hybrid Systems**
  - Morpheus, KaZaA
Napster: The P2P epic

- Founded in June 1999 by Shawn Fanning, a 19-year-old Northeastern University dropout
- Immensely successful
  - Success due to ability to create and foster an online community
  - Had 640000 users at any given moment in November 2000
  - More than 60 million people, including an estimated 73% of US students, had downloaded the software and songs in less than one year.
  - At the peak, 1 million new users per week
  - Universities begin to ban Napster, due to overwhelmed bandwidth consumption
- Battles with RIAA
  - RIAA sues Napster in Dec. 1999, asking $100K per song copied
  - Court rules Napster to shutdown in July 2001
  - files for bankruptcy in June 2002
- Technically not interesting
A brief history

June 1999
Shawn Fanning starts Napster.

Dec. 1999
RIAA sues Napster for copyright infringement.

July 2001
Napster is shut down!

Napster clones
- KaZaA
- Gnutella
- Morpheus
- MojoNation
- ....

Academic Research
- SETI@home
- folding@home
- ....
- Distributed computing

Napster clones
- CAN
- Chord
- Pastry
- Tapestry
- Hypercub
- Skip graphs
- ....

Distributed computing
Main Challenge: Locating and Routing

- How to get to another node to find a particular piece of data, considering the following:
  - Dynamics: Node can join and leave any time
  - Scalability: There may be millions of hosts
P2P with a centralized server

- Centralized design, with bottlenecks and vulnerability
- Simple implementation
- Low scalability

Napster-Server

Client

Index Table

Peer 1 has ...
Peer 2 has ...
Peer 3 has ...

register

140.112.106.24 has

140.112.106.24 has

I have “love.mp3”

Who has “love.mp3”?

download
Purpose:
- uses idle CPU cycles on ordinary PCs for massively parallel analysis of extraterrestrial radio signals

Architecture:
- central SETI@home server distributes data, similar to Napster
- analyses done locally by SETI@home screen saver
Hybrid Structure: KaZaA, Morpheus

- Has a centralized server that maintains user registrations, logs users, into the systems to keep statistics, provides downloads of client software, and bootstraps the peer discovery process.

- Two types of peers:
  - Supernodes (fast CPU + high bandwidth connections)
  - Nodes (slower CPU and/or connections)

- Supernodes addresses are provided in the initial download. They also maintain searchable indexes and proxies search requests for users.
Hybrid Structure: KaZaA, Morpheus
Supernodes act as regional index servers.
They communicate by broadcasting.

- Supernodes act as regional index server
- They communicate by broadcasting
Some Details

- On initial registration, the client may be provided with a list of more than one supernode.
- Supernodes are "elected" by the central server – users can decline.
- Supernodes can come and go so links may fail over time.
- File transfers use http protocol and port 1214 (the KaZaA port).
KaZaA - Consequences

- Huge bandwidth Hog
- Potential for original client and/or files download to be a Trojan.
  - KaZaA clients come complete with a Trojan from Brilliant Digital Entertainment.
  - 3D advertising technology + node software that can be controlled by Brilliant Digital.
  - Intent is to use the massed horsepower to host and distribute content belonging to other companies for a fee.
  - With the user’s permission of course - opt out basis (not opt in!).
  - Content to include advertising, music, video – anything digital.
  - Also have mentioned tapping unused cycles to do compute work.
Kazaa Survives By Legal Maneuvering

- **March 2001**, Kazaa is founded by two Dutchmen, Niklas Zennstrom and Janus Friis in a company called Computer Empowerment.
- The software is based upon their FastTrack P2P Stack, a proprietary algorithm for peer-to-peer communication.
- Kazaa licenses FastTrack to Morpheus and Grokster.
- **Oct. 2001** MPAA and RIAA sue Kazaa, Morpheus and Grokster.
- **Nov. 2001**, Consumer Empowerment is sued in the Netherlands by the Dutch music publishing body, Buma/Stemra. The court orders KaZaA to take steps to prevent its users from violating copyrights or else pay a heavy fine.
- **Jan. 2002**, Zennstrom&Friis sell Kazaa software and website to Sharman Networks, based in Vanuatu, an island in the Pacific, but operating out of Australia.
- **Feb. 2002**, Kazaa cuts off Morpheus clients from FastTrack.
- **April 2002**, Sharman Networks agrees to let Brilliant Digital bundle their own stealth P2P application called AltNet within KaZaA. This network would be remotely switched on, allowing KaZaA users to trade Brilliant Digital content throughout FastTrack.
Industry Countermeasures

At any time, 3 million people are using KaZaA, sharing 500 million files (October 2002)

MPAA and RIAA are using 3 “countermeasures” to stop this:

- Sue the network operators/software creators out of existence
  - Napster, Scour, Aimster, Audio Galaxy, Grokster/Morpheus/KaZaA, Replay TV…

- Berman Bill Style “Self help”
  - Denial of service attacks against network
    - Insert bogus files (Universal and Overpeer)
    - Falsify replies and misdirect queries
  - Denial of Service attacks against users
    - Overload particular nodes
Industry Countermeasures (cont.)

- Sue end users and ISPs
  - Danish Anti Pirat Gruppen issued fines up to $14,000 to approximately 150 KaZaA and eDonkey users for uploading copyrighted material.
  - In the US, civil penalties are min $750 per song, with criminal penalties as high as $250,000 and five years in prison under NETA.
  - ISPs can be sued under DMCA (The Digital Millenium Copyright Act), unless they notify offending subscribers and take down infringing material.
    - MPAA sent 54,000 cease and desist letters to ISPs in 2001, should be 100,000 in 2002. 90% result in ISPs taking action.
Big Music May Be Watching You

- MPAA uses services like Ranger Online to:
  - Identify users
    - By collecting IP addresses of uploaders and downloaders
  - Identify the content users are sharing
  - Collect evidence for notice and takedown
    - By downloading file, logging time and location
Fully Distributed P2P Systems
Case Study

- Freenet
- Gnutella
- CAN
- CFS/Chord
- OceanStore/Tapestry
Freenet

- A project led by Ian Clarke in his 4th Year Project at University of Edinburgh
- Philosophy
  - One should be able to publish and obtain information on the Internet without fear of censorship
- The result
  - A Distributed Anonymous Information Storage and Retrieval System
Design Goals

- Anonymity for both producers and consumers of information
- Deniability for stores of information
- Resistance to attempts by third parties to deny access to information
- Efficient dynamic storage and routing of information
- Decentralization of all network functions
Protocol Detail: Retrieving Data

Basically a depth-first search!
Protocol Detail: Messages

- Transaction ID
  - Randomly-generated 64-bit
- IP address and port number
- Hops-to-live
  - Decremented at each hop to prevent indefinite routing
  - Messages will still be forwarded with some probability even though hops-to-live value has reached 1, so as to prevent attacker from tracing the requester's location.
- Depth counter
  - Incremented at each hop to let a replying node to set hops-to-live high enough to reach a request.
  - Requestors should initialize it to a small random value to obscure their location.
A description of the file to be requested (e.g., its file name) is hashed to obtain a file key that is attached in the request message.

A node receiving a request for a file key checks if its own store has the file. If so, it returns the file. Otherwise, it forwards the request to the node in its routing table that has the most ‘similar’ key to the requested one.

If that node cannot successfully (and recursively!) find the requested file, then a second (again, according to key similarity) candidate from the routing table is chosen, and so on, until either the file is found, or a request failed message is returned.
Neighbor selection in routing (cont.)

- Once a copy of the requested file is found, it will be sent to the requester along the search path. Each node in the path caches a copy of the file, and creates a new entry in its routing table associating the actual data source with the requested key.
Some Observations

- Quality of routing should improve over time
  - Nodes should come to specialize in locating similar keys.
  - Nodes should become similarly specialized in storing files having similar keys.
- Popular data will be replicated and mirrored closer to requesters.
- Connectivity increases as requests are processed.
- Note that files with similar hashed keys do not mean that they are similar in content.
  - A crucial node failure cannot cause a particular subject to extinguish.
Storing Data

- Newly inserted files are selectively placed on nodes already possessing files with similar keys.
- Insert basically follows the same depth-first like search to see if there is a key collision. If so, the colliding node returns the pre-existing file as if a request has been made. If no collision, then the file will be placed on each node in the search path.
  - The nodes will also update their routing tables, and associate the inserter as the data source with the new key.
  - For security reason, any node along the way can unilaterally decide to change the insert message and claim itself or another arbitrarily node as the data source.
Observations

- Newly inserted files are selectively placed on nodes already possessing files with similar keys.
- New nodes can use inserts as a supplementary means of announcing their existence to the rest of the network.
- An attempt to supply spurious files will likely simply spread the real file further, as the original file is propagated back on collusion.
Managing Data

- Node storage is managed as an LRU (Least Recently Used) cache.
- No file is *eternal*! A file could disappear in the network if it has not been requested for a long period of time.
- Inserted files are encrypted so that node operators can ‘claim’ innocence of possessing some controversial files.
  - The requester who knows the actual file name can use that information to decrypt the encrypted file.
    - Vulnerable to dictionary attack
Freenet Naming

- Hierarchical name system
  - Files are identified by the hash of their filenames
  - Cannot have multiple files with the same name
  - Global single-level namespace is not desirable, since malicious users can engage in “key-squatting”

- Two-level namespace
  - Each user has their own directory
Freenet File Export

- Consider exporting file with name "My life.mp3"
  - Compute a public/private key pair from name using a deterministic algorithm

- File is encrypted with the hash of the public key
  - Goal is not to protect data – the file contents should be visible to anyone who knows the original keyword
  - Goal is to protect site operators – if a file is stored on your system, you have no way of decrypting its contents

- File is signed with the private key
  - Integrity check (though not a very strong one)
Freenet Directories

- **Two - level directories**
  - Users can create a signed-subspace
  - Akin to creating a top-level directory per user
  - Subspace creation involves generating a public/private key pair for the user
  - The user’s public key is hashed, XORed and then rehashed with the file public key to yield the file encryption key
- **For retrieval, you need to know the user’s public key and the file’s original name**
Adding Nodes

- Node connecting to network must obtain existing node’s address through out-of-band means

- Once connected, a new node message is propagated to randomly selected, connected nodes so existing nodes learn of new node’s existence
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
Overlay

- Fully Distributed P2P Networks are essentially an overlay over the Internet
- Neighbors in the overlay may not necessarily reflect proximity
Case Study: GNUTELLA

Peer-to-Peer Storage Space Sharing System
Gnutella History

- Originally conceived of by Justin Frankel, 21 year old founder of Nullsoft
- March 2000, Nullsoft posts Gnutella to the web
- A day later AOL removes Gnutella at the behest of Time Warner
- There are multiple open source implementations
  - Jtella
  - Gnucleus
- The Gnutella protocol has been widely analyzed
GNUTELLA

- Gnutella was developed in the early 2000 by Justin Frankel’s Nullsoft. Nullsoft was later acquired by AOL, which has now stopped this project.
- But, this application had already distributed as an open source application, and so different versions still exist.
- Gnutella has become a Protocol now.
Design Goals

- Ability to operate in a dynamic environment
- Performance and Scalability
- Reliability
- Anonymity
Characteristics

- message broadcasting for node discovering and search requests (Call-and-Response protocol)
- forming of overlay network; connecting: join the "several known hosts"
- user data transfer: store and forward using HTTP
Protocol Definition

- A new node (called a **servent** in Gnutella) joins a system by connecting to a known host.
- Communication between servents:
  - A set of descriptors used for communicating data between servents
  - A set of rules governing the inter-servent exchange of descriptors
## Defined Descriptors

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ping</td>
<td>Used to actively discover hosts on the network</td>
</tr>
<tr>
<td>Pong</td>
<td>The response to a Ping</td>
</tr>
<tr>
<td>Query</td>
<td>The primary mechanism for searching the distributed network</td>
</tr>
<tr>
<td>QueryHit</td>
<td>The response to a Query</td>
</tr>
<tr>
<td>Push</td>
<td>A mechanism that allows a firewalled servent to contribute file-based data to the network</td>
</tr>
</tbody>
</table>
A new node connects to one of several known hosts.

flooding of PING/PONG messages; broadcasting range limited by TTL counter.

short time memory of messages already seen; prevents re-broadcasting; GUIDs to distinguish msg

To cope with the dynamic environment, a node periodically PINGs its neighbors to discover other participating nodes.
1) Node A asks Node B for data.

2) B keeps a record that A initiated the request.

3) B forwards the request to its neighbors.

4) They return any matching info.

5) B looks up source of request.

6) B returns matching info.

7) A may initiate download using HTTP.

- search: Query/Query-Response (flooding/breadth-first search!)
- download: GET/PUSH. (direct transmission)
Limitations/Difficulties:

- unstable/loose connectivity of the servents
  - performance management difficult
- scalability: e.g. TTL=10, every node broadcasts to six others msg;
  - problem in huge networks
- low TTL, low search horizon
- denial-of-service attacks
Free riding on Gnutella

<table>
<thead>
<tr>
<th>The Top</th>
<th>Share</th>
<th>As percent of the whole</th>
</tr>
</thead>
<tbody>
<tr>
<td>1%</td>
<td>1,142,645</td>
<td>37%</td>
</tr>
<tr>
<td>5%</td>
<td>2,182,087</td>
<td>70%</td>
</tr>
<tr>
<td>10%</td>
<td>2,692,082</td>
<td>87%</td>
</tr>
<tr>
<td>20%</td>
<td>3,037,232</td>
<td>98%</td>
</tr>
<tr>
<td>25%</td>
<td>3,082,572</td>
<td>99%</td>
</tr>
</tbody>
</table>

http://www.firstmonday.dk/issues/issue5_10/adar/
Partial Map of the Gnutella Network

Source: http://dss.clip2.com

Source: http://www.limewire.com
Gnutella Host Count - 2001

Gnutella Network Hosts

No. of hosts

Time


Hosts overall Unique hosts
Gnutella Host Count - 2003

Gnutella Network Hosts (courtesy of LimeWire.com)
Gnutella Network Analysis

Some Findings

- More than 40,000 hosts were found.
- The number of connected components is relatively small
  - The largest connected component includes more than 95% of the active nodes.
  - The 2\textsuperscript{nd} largest connected component usually has less than 10 nodes.
- The dynamic graph structure
  - 40% of the nodes leave the network in less than 4 hrs
  - 25% are alive for more than 24 hrs.
Figure 4: Average node connectivity. Each point represents one Gnutella network. Note that, as the network grows, the average number of connections per node remains constant (average node connectivity is 3.4 connections per node).

Source: [Ripeanu2001]
**Figure 5**: Connectivity distribution during November 2000. Each series of points represents one Gnutella network topology we discovered at different times during that month. Note the log scale on both axes. Gnutella nodes organized themselves into a power-law network.

Source: [Ripeanu2001]
In a power-law network, the fraction of nodes with $L$ links is proportional to $L^{-k}$, where $k$ is a network dependent constant.

- most nodes have few links and a tiny number of hubs have a large number of links.

Power-law networks are generally highly stable and resilient, yet prone to occasional catastrophic collapse.

- extremely robust when facing random node failures, but vulnerable to well-planned attacks.

The power law distribution appears in Gnutella networks, in Internet routers, in the Web, in call graphs, ecosystems, as well as in sociology.
power-law graph
AT&T Call Graph

# of telephone numbers from which calls were made

# of telephone numbers called

Aiello et al. STOC ‘00
queries broadcast to every node within radius \( ttl \)  
\( \Rightarrow \) as network grows, encounter a bandwidth barrier  
(dial up modems cannot keep up with query traffic, fragmenting the network)
Content-Addressable Network (CAN)
**CAN: overview**

- A lookup protocol (indexing system) that maps a desired key to a value
  - insert (key, value)
  - value = retrieve (key)
    - Example: key can be a hashed value of a file name, and key be the IP address of the host storing the file.

- A storage protocol layered on top of the lookup protocol then takes care of storing, replicating, caching, retrieving, and authenticating the files.

- Uses virtual $d$-dimensional coordinate space on a $d$-torus to store (key, value) pairs, and a uniform hash function to map keys to points in the $d$-torus

- At any instant the entire coordinate space is partitioned among all the peers in the system; each peer “owns” one individual, distinct zone.
  - (key, value) pair of a file is stored at the peer that owns the zone containing the corresponding point of the key.
Example: 2-D CAN

When host 140.112.106.12 wishes to insert a file “love.mp3” into the system, it hashes the name to obtain a key, say (0100,1100).

Then the entry (0100,1100), 140.112.106.2 ) is added to node y’s index table.

index information for file “love.mp3”, i.e., ( (0100,1100), 140.112.106.2 )

To find file “love.mp3”, one uses the same hash function to find the key (0100,1100); then, routes to the peer handling the point/key (0100,1100), which is y, and then obtains the IP 140.112.106.2 from y.

So, inserting and finding a file becomes a routing problem: how to find a peer handling a given point in the coordinate space?
Routing in CAN

- Intuitively – following the straight path through the Cartesian space from source to destination
- Node maintains coordinate routing table that holds IP addresses and zones’ coordinate of its neighbors in the space
- Two nodes are neighbors if their coordinate spans overlap along $d−1$ dimensions and abut along one dimension
- CAN message contains destination coordinate. Node greedy forwards it to the neighbor with coordinates closest to the destination coordinate
Routing in CAN

Neighbors of node 12: 8, 10, 11, 13, 16
Routing Complexity

- For the d-dimensional space equally partitioned into n nodes the average routing path is \((d/4)*n^{(1/d)}\)
- Individual nodes maintain 2d neighbors
- The path length growth proportionally to the \(O(n^{(1/d)})\)
- Many different routes between two points
Basic operations of CAN

- Inserting, updating, deleting of (key,value) pairs
- Retrieving value associated with a given key
- Adding new nodes to CAN
- Handling departing nodes
- Dealing with node failures
**CAN construction**

- New node is allocated its own portion of the coordinate space in three steps:
  - Find a node already in the CAN – look up the CAN domain name in DNS
  - Pick zone to join to and route request to its owner using CAN mechanisms
  - Split the zone between old and new node
  - The neighbors of the split zone must be notified so the routing can include the new node
Finding a zone

- The new peer X finds a node E to join the network
- X then finds a peer whose zone will be split.
  - X Randomly chooses a joining point J for load balancing
  - X sends a JOIN request message destined for point J (handled by P) via E using CAN routing mechanisms
Splitting a zone

- P splits its zone in half and assigns one half to X (J), assuming certain ordering of the dimensions, i.e. first X then Y.
- Transfer (key, value) pairs from the half of the zone to the new node.
Joining the Routing

- The neighbors of the split zone must be notified so that routing can include the new node.
  - O sends its routing table to N (IPs and Zones of O’s neighbors)
  - Having O’s routing table, N may figure out its routing table.
  - O re-computes its routing table.
  - Both O’s and N’s neighbors must be informed of this reallocation of space.
  - All of their neighbors update their own routing table.
Joining the Routing: Illustration

Before 21 joins:
9: 4, 8, 10, 13
4: 3, 5, 9
8: 2, 3, 7, 9, 12
10: 5, 9, 14
13: 9, 12, 14, 17

After 21 joins:
9: 4, 8, 21, 13
4: 3, 5, 9, 21
8: 2, 3, 7, 9, 12
10: 5, 21, 14
13: 9, 12, 14, 17, 21
21: 4, 9, 10, 13
Node departure and recovery

- **Normal procedure** – explicit hand over of (key,value) database to one of the neighbors
- **Node failure** – immediate takeover procedure:
  - Failure detected as a lack of update messages
  - Each neighbor starts timer with proportion to the node’s zone size
  - After timer expires the node extends its own zone to contain the failed neighbor’s zone and sends TAKEOVER message to all failed node’s neighbors
  - On receive of the TAKEOVER node cancels its timer if the sender’s zone size is smaller than his own. Otherwise it sends it’s own TAKEOVER message.
Recovery cont.

- CAN state may become inconsistent if multiple adjacent nodes fail simultaneously
- In such cases perform an expanding ring search for any nodes residing beyond the failure region (?)
- If it fails initiate repair mechanism (?)
- If a node holds more than one zone initiate the background zone-reassignment algorithm
Zone Reassignment

- **Case 1:** when zone 2 is to be merged
  - search for 2’s sibling --- node 3
  - node 3 takes over zone 2.

```
  1  3  8
  4  6  9 10
  5  7  9 11
```

Binary Partition tree
Zone Reassignment (cont.)

- Case 2: when zone 9 is to be merged
  - Step 1: Searching for sibling of 9, but fail
  - Step 2: Use DFS until two sibling leaves are reached
  - Step 3: Merge zone 10 with zone 11 and takeover by node 11
  - Step 4: node 11 now takeover zone x, DONE!!
MIT Chord

A Scalable Peer-to-peer Lookup Protocol for Internet Applications
Chord

- Chord provides support for just one operation: given a key, it maps the key onto a node.
- Applications can be easily implemented on top of Chord.
  - Cooperative File System (CFS)
  - DNS
Chord-based distributed storage system: CFS
The Block Store Layer

- A CFS File Structure example
- The root-block is identified by a public-key and signed by corresponding private key
- Other blocks are identified by cryptographic hashes of their contents
Chord properties

- Efficient: $O(\log N)$ messages per lookup
  - $N$ is the total number of servers
- Scalable: $O(\log N)$ state per node
- Robust: survives massive changes in membership
Hashing

- Hashing is generally used to distribute objects evenly into a set of servers
  - E.g., the linear congruential function $h(x) = ax + b \pmod{p}$
  - SHA-1

- When the number of servers changes ($p$ in the above case), then almost every item would be hashed to a new location
  - Cached objects become useless in each server when a server is removed or introduced to the system.

![Diagram showing hash function and bucket management](image-url)
Consistent Hashing

- Load is balanced
- Relocation is minimum
  - When an $N$th server joins/leaves the system, with high probability only an $O(1/N)$ fractions of the data objects need to be relocated
A possible implementation

Objects are servers are first mapped (hashed) to points in the same interval.

Then objects are actually placed into the servers that are closest to them w.r.t. the mapped points in the interval.

- E.g., D001→S0, D012→S1, D313→S3
When server 4 joins

Only D103 needs to be moved from S3 to S4. The rest remains unchanged.
When server 3 leaves

Only D313 and D044 need to be moved from S3 to S4.
Consistent Hashing in Chord

- Node’s ID = SHA-1 (IP address)
- Key’s ID = SHA-1 (object’s key/name)
- Chord views the ID’s as
  - Uniformly distributed
  - occupying a circular identifier space
- Keys are placed at the node whose IDs are the closest to the (ids of) the keys in the clockwise direction.
  - $\text{successor}(k)$: the first node clockwise from $k$.
  - Place object $k$ to $\text{successor}(k)$.
An ID Ring of length $2^6 - 1$
Simple Lookup

- Lookup correct if successors are correct
- Average of \( n/2 \) message exchanges

Circular ID Space

lookup(k54)
The $i$th entry in the finger table points to $\text{successor}(n+2^{i-1} \ (\text{mod} \ 2^6))$
Scalable Lookup

- Look in local finger table for the largest $n$ s.t. $\text{my\_id} < n < \text{key\_id}$
  - If $n$ exists, call $n\_lookup(key\_id)$, else return $\text{successor(my\_id)}$
Scalable Lookup

Finger table at N42

<table>
<thead>
<tr>
<th>Pointer</th>
<th>Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>N42+1</td>
<td>N48</td>
</tr>
<tr>
<td>N42+2</td>
<td>N48</td>
</tr>
<tr>
<td>N42+4</td>
<td>N48</td>
</tr>
<tr>
<td>N42+8</td>
<td>N51</td>
</tr>
<tr>
<td>N42+16</td>
<td>N1</td>
</tr>
<tr>
<td>N42+32</td>
<td>N14</td>
</tr>
</tbody>
</table>

lookup(k54)
Scalable Lookup

Each node can forward a query at least halfway along the remaining distance between the node and the target identifier.

Lookup takes $O(\log N)$ steps.
Node joins

- When a node $i$ joins the system from any existing node $j$:
  - Node $j$ finds $\text{successor}(i)$ for $i$, say $k$
  - $i$ sets its successor to $k$, and informs $k$ to set its predecessor to $i$.
  - $k$'s old predecessor learns the existence of $i$ by running, periodically, a stabilization algorithm to check if $k$'s predecessor is still $i$. 
Node joins (cont.)

Circular ID Space

N25 joins via N8

Finger table

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>N8+1</td>
<td>N14</td>
</tr>
<tr>
<td>N8+2</td>
<td>N14</td>
</tr>
<tr>
<td>N8+4</td>
<td>N14</td>
</tr>
<tr>
<td>N8+8</td>
<td>N21</td>
</tr>
<tr>
<td>N8+16</td>
<td>N32</td>
</tr>
<tr>
<td>N8+32</td>
<td>N42</td>
</tr>
</tbody>
</table>

✗ aggressive mechanisms requires too many messages and updates
Node Fails

- Can be handled simply as the invert of node joins; I.r., by running stabilization algorithm.
Handling Failures

- Use successor list
  - Each node knows $r$ immediate successors
  - After failure, will know first live successor
  - Correct successors guarantee correct lookups

- Guarantee is with some probability
  - Can choose $r$ to make probability of lookup failure arbitrarily small
Weakness

- NOT that simple (compared to CAN)
- Member joining is complicated
  - aggressive mechanisms requires too many messages and updates
  - no analysis of convergence in lazy finger mechanism
- Key management mechanism mixed between layers
  - upper layer does insertion and handle node failures
  - Chord transfer keys when node joins (no leave mechanism!)
- Routing table grows with # of members in group
- Worst case lookup can be slow
Chord Summary

- **Advantages**
  - Filed guaranteed to be found in $O(\log(N))$ steps
  - Routing table size $O(\log(N))$
  - Robust, handles large number of concurrent join and leaves

- **Disadvantages**
  - Performance: routing in the overlay network can be more expensive than in the underlying network
    - No correlation between node ids and their locality; a query can repeatedly jump from Taiwan to America, though both the initiator and the node that store the item are in Taiwan!
  - Partial solution: Weight neighbor nodes by Round Trip Time (RTT)
    - when routing, choose neighbor who is closer to destination with lowest RTT from me » reduces path latency
Related Work: OceanStore

A Global Scale Persistent Storage Utility Infrastructure
OceanStore: Motivation

- Ubiquitous Computing
  - Computing everywhere,
  - Connectivity everywhere
- But, are data just out there?
- OceanStore: An architecture for global-scale persistent storage

[Source: http://oceanstore.cs.berkeley.edu/]
Challenges

- Magnitude
  - Assume $10^{10}$ people in world, say 10,000 files/person (very conservative?), then $10^{14}$ files in total!
  - If 1MB/file, then $10^{20}$ size is needed
  - Surely, this must be maintained cooperatively by many ISPs.

- Persistent
  - Geographic independence for availability, durability, and freedom to adapt to circumstances
Challenges (cont.)

- **Security**
  - Encryption for privacy, signatures for authenticity, and Byzantine commitment for integrity

- **Robust**
  - Redundancy with continuous repair and redistribution for long-term durability

- **Management**
  - Automatic optimization, diagnosis and repair

- **Anti-trust**
  - Utility Infrastructure
  - Users pay monthly fee to access their data
Design Goals

- **Untrusted Infrastructure:**
  - The OceanStore is comprised of untrusted components
  - Only ciphertext within the infrastructure

- **Nomadic Data: data are allowed to flow freely**
  - **Promiscuous Caching:** Data may be cached anywhere, anytime
    - continuous *introspective monitoring* is used to discover tacit relationships between objects.
  - Optimistic Concurrency via Conflict Resolution
The core of the system is composed of a multitude of highly connected "pools", among which data is allowed to "flow" freely. Clients connect to one or more pools, perhaps intermittently.

[Source: http://oceanstore.cs.berkeley.edu/]
Secure Naming

- Unique, location independent identifiers:
  - Every *version* of every unique entity has a permanent, *Globally Unique ID (GUID)*
  - 160 bits SHA-1 hashes
    - $2^{80}$ names before name collision

- Naming hierarchy:
  - Users map from names to GUIDs via hierarchy of OceanStore objects (*ala SDSI*)
  - Requires set of “root keys” to be acquired by user
Data Location and Routing

- a two-tiered approach
  - Attenuated bloom filters
    - fast, probabilistic
  - Wide-scale distributed data location
    - slower, reliable, hierarchical
Bloom Filter (BF)

- A probabilistic algorithm to quickly test membership in a large set using multiple hash functions into a single array of bits.

Main Idea:

\[ h(x) \neq h(y) \Rightarrow x \neq y \]
\[ h(x) = h(y) \nRightarrow x = y \]
Bloom Filter Design

- Use an $m$ bits vector, initialized to 0’s, for the BF.
  - Larger $m$ => fewer filter errors.
- Choose $h > 0$ hash functions: $f_1(), f_2(), \ldots, f_h()$.
- When key $k$ inserted into DB, set bits $f_1(k), f_2(k), \ldots, f_h(k)$ in the BF to 1.
  - $f_1(k), f_2(k), \ldots, f_h(k)$ is the signature of key $k$. 
Example

- \( m = 11 \) (normally, \( m \) would be much larger).
- \( h = 2 \) (2 hash functions).
- \( f_1(k) = k \mod m \).
- \( f_2(k) = (2k) \mod m \).

- K=15 inserted
- K=17 inserted
Example

- DB has k = 15 and k = 17.
- Search for k.
  - $f_1(k) = 0$ or $f_2(k) = 0 \Rightarrow k$ not in DB.
  - $f_1(k) = 1$ and $f_2(k) = 1 \Rightarrow k$ may be in DB.

- k = 25 => not in DB
- k = 6 => filter error.
Bloom Filter Design

- Choose $m$ (filter size in bits).
  - Use as much memory as is available.
- Pick $h$ (number of hash functions).
  - $h$ too small $\Rightarrow$ probability of different keys having same signature is high.
  - $h$ too large $\Rightarrow$ filter becomes filled with ones too soon.
- Select the $h$ hash functions.
  - Hash functions should be relatively independent.
Attenuated bloom filters

- An attenuated bloom filter of depth D can be viewed as an array of D normal Bloom filters
  - The first Bloom filter is a record of the objects contained locally on the current node.
  - The $i^{th}$ Bloom filter is the union of all of the Bloom filters for all of the nodes a distance $i$ through any path from the current node.

- Query is routed along the edges whose filters indicate the presence of the object at the smallest distance.
The probabilistic query process in OceanStore

Key (0,1,3)

11100
11011

Local BF

n1

10101

11100
11011

1st BF

2nd BF

n2

11100

00011

n4

11010

n3

Key (0,1,3)

00011

11010

Lookup at n1

Local BF

i^{th} BF
Tapestry

- A prototype of a decentralized, fault-tolerant, adaptive overlay infrastructure
- Network substrate of OceanStore
  - Routing: Suffix-based hypercube
    Similar to Plaxton, Rajamaran, Richa (SPAA97)
  - Decentralized location:
    Virtual hierarchy per object with cached location references
Routing and Location

- **Object location:**
  - map GUIDs to root node IDs
    - Each object has its own hierarchy rooted at Root
    - Root responsible for storing object’s location

- **Suffix routing from A to B**
  - At \( h^{th} \) hop, arrive at nearest node hop(h) such that:
    - hop(h) shares suffix with B of length \( h \) digits
  - Example: 5324 routes to 0729 via
    - \( 5324 \rightarrow 7342 \rightarrow 3429 \rightarrow 4729 \rightarrow 0729 \)
Basic Plaxton Mesh
Incremental suffix-based routing

NodeID 0x79FE
NodeID 0x44FE
NodeID 0x35E
NodeID 0x40E
NodeID 0x23FE
NodeID 0xA1FE
NodeID 0x73FF
NodeID 0x423E
NodeID 0x239E
NodeID 0x1290
NodeID 0xF990
NodeID 0x993E
NodeID 0x9990
NodeID 0x43FE
NodeID 0x13FE
NodeID 0x1290
NodeID 0x04FE
NodeID 0x555E
NodeID 0x035E
NodeID 0x73FF
NodeID 0x239E
NodeID 0x1290
NodeID 0xF990
NodeID 0x993E
NodeID 0x9990
NodeID 0x43FE
# Neighbor Table

Neighbor table at \((3120)_4\)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0120</td>
<td>1120</td>
<td>2120</td>
<td>3120</td>
</tr>
<tr>
<td>2</td>
<td>x020</td>
<td>x120</td>
<td>x220</td>
<td>x320</td>
</tr>
<tr>
<td>1</td>
<td>xx00</td>
<td>xx10</td>
<td>xx20</td>
<td>xx30</td>
</tr>
<tr>
<td>0</td>
<td>xxx0</td>
<td>xxx1</td>
<td>xxx2</td>
<td>xxx3</td>
</tr>
</tbody>
</table>
Surrogate Routing

- Neighbor table will have holes
  - Must be able to find unique root for every object
  - Tapestry’s solution: try next highest.
Node Joins

- A little bit complicated
  - Try to think about the problem by yourself
BitTorrent

- Official Website: http://bittorrent.com/
- Created by Bram Cohen in 2002
- Ideal for distributing large (gigabyte) files
Traditional File Sharing Method: Client Server Architecture

Source is the bottleneck!
Freenet style P2P Sharing

- It helps distribute files; but, still, request is kind of client-server style
- A client can help distribute the content only when it has completed the downloading
Bit Torrent

- A peer helps others to distribute the file (by uploading the part of the file it has downloaded) while it is downloading.

We need to know who (downloading) have what
In BT, this information is centralized at trackers.
Some terms in BT

- **Torrent files [ *.torrent ]** are static metadata files containing information such as file size, name, checksum, and the IP of a tracker.
- **Downloaders (leechers):** Peers that have only a part (or none) of the file.
- **Seeds:** Peer has the complete file, and chooses to stay in the system to allow other peers to download.
- **Tracker:** Coordinating peer
Piece

- A file is cut into pieces of fixed size, typically 256Kb
- Each downloader reports to all of its peers what pieces it has.
- To verify data, Hash codes are used for all the pieces, included in .torrent files
- Selecting pieces to download in a good order is very important for good performance.
  - **Rarest First**: Peers generally download pieces with the fewest number of owners.
  - Slow transfer rate from some peers may delay a download’s finish.
    - **Endgame Mode**: If a download is close to its end, it will send request to all peers for the missing pieces.
Choking

- “Choking” is to temporary refuse to upload to someone (but download can continue)
- A BT peer always unchokes a several number of other peers (default: 4), so the issue is which peer to unchoke.
- BT’s choking algorithms use “tit-for-tat” strategy: the more you upload, the more you download
  - Free riders will be `choked`
  - Many more interesting strategies to study
Optimistic Unchoking

- A BitTorrent peer has a single ‘optimistic unchoke’ which is uploaded regardless of the current download rate from it. This peer rotates every 30s.

Reason:
- To discover if currently unused connections are better than the ones being used.
P2P vs. Copyright

- P2p disrupts traditional distribution mechanisms
- Notions of copyright and intellectual property need to be put in a digital-age context (and new business models will need to be developed and implemented)

- 科技與智慧財產權的衝突
- 新技術的發明必定會與舊的經濟體系（既得利益者）產生衝突
- 談判、協議、合作，進而形成新的商業模式，並且由法律追認。
Research Problems

- Overlay constructions
  - other topologies of structured P2P networks
    - Koorde
    - Kademlia
  - hybrid structures (structured + unstructured)

- Search
  - Keyword Search
  - Prefix Search
Koorde

- A binary ID $b_{k-1}b_{k-2}...b_1b_0$ connects to two nodes $b_{k-2}...b_1b_00$ and $b_{k-2}...b_1b_01$
- Search can be done on $O(\log N)$ steps, using only $O(1)$ space

How to get to 110111 from 011001?
Kademlia

- A binary ID $b_{k-1}b_{k-2}...b_1b_0$ must know at least one node of ID $b_{k-1}b_{k-2}...b_1b_0$, $b_{k-1}b_{k-2}...b_1b_0$, and so on.

110 looks for 010
Classification of P2P

- **Centralized Management**
  - Napster, Seti@Home

- **Fully Distributed Systems**
  - Unstructured:
    - Gnutella, Freenet
  - Structured (Distributed Hash Tables, DHT)
    - CFS/Chord, Tapestry, Pastry

- **Hybrid Systems (centralized+distributed)**
  - Morpheus, KaZaA
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>Simplicity</th>
<th>Robustness</th>
<th>Fault Tolerance</th>
<th>Salability</th>
<th>Search Flexibility</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Centralized</strong></td>
<td>+++</td>
<td>--</td>
<td>-</td>
<td>-</td>
<td>All kinds</td>
</tr>
<tr>
<td><strong>Unstructured</strong></td>
<td>++</td>
<td>++</td>
<td>+++</td>
<td>+</td>
<td>All kinds, but not guaranteed</td>
</tr>
<tr>
<td><strong>DHT</strong></td>
<td>-</td>
<td>+</td>
<td>++</td>
<td>+++</td>
<td>Exact name match</td>
</tr>
<tr>
<td><strong>Hybrid</strong></td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>All kinds, but not guaranteed</td>
</tr>
</tbody>
</table>

Types of Search: name match, keyword search, range query, wildcard search, …
Two Types of Name Services

- **White page service**
  - search by names
  - “Lord of the rings.mpg”

- **Yellow page service**
  - search by attributes
  - “rings”, “lord”, “mpg”
  - **Keyword search** is the basis for yellow page services

- Both services can be easily supported in unstructured P2Ps or P2Ps with a centralized server. Yellow page service, however, is not easy in DHTs.
Common Technique: Inverted Index
Distributed Inverted Indexing

- $w_1 \{A, B, D\}$
- $w_2 \{A, C, E\}$
- $w_3 \{B, E\}$
- $w_4 \{C, E\}$
- $w_5 \{B, D\}$

Keywords $= \{W_1, W_5\}$
Zipf's law

- In a real world corpus, keyword frequency—the count of a keyword's occurrence in objects—varies enormously. A few keywords occur very often while many others occur rarely (in power-law relationship).
  - e.g., mp3, ring, lord
- Zipf’s law implies that a straightforward distributed implementation of inverted index results in an extremely imbalanced load.
Other Problems

- **Storage redundancy**
  - an object \( o \) contains keywords \( \{w_1, w_2, \ldots, w_k\} \) is repeatedly stored at \( k \) different sites.
    - Increase insert/delete complexity
    - Decrease consistency

- **Fault tolerance**
  - A failure to a site would block all queries containing a keyword handled by the site.
  - Nodes handling hot keywords may be swamped.

- **Object ranking is difficult**
  - Ranking in general requires global knowledge
    - *inverse document frequency* (IDF)
Prefix Search

- Human readable object names often inherit some meaningful ordering
  - “Lord of the rings-I.mpg”,
  - “Lord of the rings-II.mpg”
  - “Lord of the rings-III.mpg”
  - “HP LJ2200d”, “HP LJ1200”, “HP LJ5500”

- **Prefix search (range query)** is therefore useful and convenient in retrieving related objects
  - “Lord of the rings* ”
  - “HP LJ*”
Prefix Search is Difficult!

- Load imbalance
- Increase search cost
- Require global knowledge
Taking Proximity into Consideration

(a) An initial empty system.

(b) Three nodes join the system and claim their IDs.

(c) More nodes join the system.
Proximity causes some conflicts with load balanced prefix search
Content Delivery Network

content provider

Internet
Motivations

- Increasing Internet traffic
- Congestion in the Internet.
- Web Servers sometimes become overloaded due to too many people trying to access their content over a short period of time.
  - 911 Attack
- The 8-Second Rule
  - If your webpage hasn't loaded within 8 seconds, chances are your viewers are history.
- Complexity and Cost of managing and operating a global network
- Increasing demand of rich content delivery
What is a Content Delivery Network?

- Network of content servers deployed throughout the Internet available on a subscription basis to publishers.
- Web publishers use these to store their high-demand or rich content (i.e., certain portions of their web site).
- Support for delivery of many content types (e.g., HTML, graphics, streaming media, etc.)
- Brings content closer to end-users but no changes required at end-hosts.
CDN: How does it work?

**Preparation:**
- Web publishers decide on the portions of their web site they want to be served by the CDNs.
  - Use CDNs for images or rich content.
  - Most web pages: 70% objects
- CDN companies provide web content distributors with the software tools to modify their HTML code.
- CDN (e.g., Akamai) creates new domain names for each client content provider.
  - e.g., a128.g.akamai.net
- The URL's pointing to these objects on the publishers server are then modified so that the content can now be served from the CDN servers.
  - "Akamaize" content
    - http://www.cnn.com/image-of-the-day.gif becomes
    - http://a128.g.akamai.net/image-of-the-day.gif
- Using multiple domain names for each client allows the CDN to further subdivide the content into groups.
  - DNS sees only the requested domain name, but it can route requests for different domains independently.
Akamai with DNS hooks

www.cnn.com
“Akamaizes” its content.

DNS server for cnn.com

get http://www.cnn.com

“Akamaized” response object has inline URLs for secondary content at a128.g.akamai.net and other Akamai-managed DNS names.

akamai.net DNS servers

lookup a128.g.akamai.net

Akamai servers store/cache secondary content for “Akamaized” services.

Source: Jeff Chase
CDN: How does it work?

- Monitoring/Routing:
  - Some kind of probing algorithms used to monitor state of network - traffic conditions, load on servers, and location of users.
  - Generate network map incorporating this information - maps updated frequently to ensure the most current view of the network.
  - CDN develops its own “routing tables to direct the user to the fastest location.”
CDN: How does it work?

- **Delivery:**
  - Data to be served by CDNs is pre-loaded onto the servers.
  - CDNs take care of migration of data to the appropriate servers.
  - Users retrieve modified HTML pages from the original server, with references to objects pointing to the CDN.
  - Content is served from the best server.
**CDN Benefits:**

- **Highly scalable:**
  - As the demand for a document increases, the number of servers serving that document also increases.
  - Ensure that no content server is overloaded by requests.

- **Fault Tolerant: guarantee 100% uptime**

- **High speed connections from content servers to the Internet.**
CDN and Layer 4 Switching:

- What is Layer 4 switching?
  - Switch employs the information contained in the transport header to assist in switching traffic.
  - Layer 4 info - port numbers to identify applications (port 80 for HTTP, 20/21 for FTP, etc.)
- Switch keeps track of established sessions to individual servers
  - use Destination IP address + destination port + Source IP address + source port for session identification
CDN and Layer 4 Switching:

- Switch performs Load Balancing:
  - Multiple servers assigned the same virtual IP address.
  - Switch maintains information on server loads.
  - Traffic load-balancing done based on specified criteria (e.g., least connections, round robin, etc.).
- Maintain session management information:
  - Ensure that all packets within a session are forwarded to the same server.
  - Ex: eShopping sessions: 2 connections - persistent HTTP for shopping cart and SSL for purchases within cart.
Akamai: Service Operator Model

- **Background**
  - Founded by a group of MIT students and professors in 1998

- **Biz-Model**
  - Customer is content provider
  - Value: quality of service
    - Given contents provided by clients, Akamai deploys the contents into its worldwide network to provide some “quality of service” browsing

- **Network Deployment**
  - More than 15,000+ servers deployed in over 1,100+ networks in 66+ countries
Some Theoretical Foundations

- Consistent Hashing and Random Trees [Karger, et al., STOC 1997]
Random Trees

- Use a tree of caches to coalesce requests
- Balance load by using a different tree for each page and assigning tree nodes to caches via a random hash function.
Random Trees (cont.)

- All requests for pages is a 4-tuple:
  - Requester’s Id
  - Name of the desired page
  - A routing path
  - A sequence of caches that should act as the nodes in the path
  - The root of the tree is always the server for the requested page
  - All other nodes are mapped to the caches by a hash function
    \[ h: P \times [1..C] \rightarrow C, \text{ where} \]
    \[ P: \text{the set of all pages} \]
    \[ C: \text{the number of caches} \]
    \[ C: \text{the set of caches} \]

- A cache stores a copy of a requested page only after it has seen \( q \) requests for the page.
Random Trees (cont.)

- **Browsing:**
  - A browser picks a random leaf to root path, maps the nodes to machines with $h$, and asks the leaf node for the page.

- **Cache:**
  - When a cache receives a request, returns a copy of the page if it has it, or forwards the request to the next node. It also increments a counter for the page and the node it is acting as.
Random Trees in an Inconsistent World

- In the above scheme, a consistent hash function can be used for $h$ to cope with the dynamic nature of the cache servers.
- A node can also know only about a $1/t$ fraction of the cache servers without a risk (in reasonable probability) of swamping or load unbalancing.