A Survey of Peer-to-Peer Content Distribution Technologies

Stephanos Androutsellis-Theotokis and Diomidis Spinellis
ACM Computing Surveys, December 2004

Presented by Eunsang Cho
Outline

- **Introduction**
  - Definition of Peer-to-Peer Systems
  - Classification of P2P Applications
  - P2P Content Distribution

- **Analysis Framework**

- **Peer-to-Peer Distributed Object Location and Routing**

- **Other Aspects**

- **Conclusion**
Definition of Peer-to-Peer Systems

- Peer-to-peer systems are distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server of authority.
Classification of P2P Applications (1/2)

• Communication and Collaboration
  – Direct, usually real-time, communication and collaboration between peer computers
  – E.g., chat and instant messaging: IRC, MSN

• Distributed Computation
  – To take advantage of the available peer computer processing power
  – E.g., Seti@home, genome@home
Classification of P2P Applications (2/2)

• Internet Service Support
  – Supporting a variety of Internet services
  – E.g., P2P multicast systems, Internet indirection infrastructures

• Database Systems
  – Designing distributed database systems
  – E.g., Local Relational Model, PIER, Piazza system

• Content Distribution
  – Sharing of digital media and other data between users
  – E.g., Napster, Publius, Gnutella, Freenet
P2P Content Distribution

• P2P Applications
  – P2P “file exchange” systems
    • Typically light-weight applications with best-effort approach
  – P2P content publishing and storage systems
    • Creating a distributed storage medium in a secure and persistent manner

• P2P Infrastructures
  – Routing and location
    • Efficiency and fault tolerance
  – Anonymity
    • Explicit aim of providing user anonymity
  – Reputation management
    • Reputation information must be kept secure, up-to-date, and available in the various network nodes.
Outline

• Introduction
• Analysis Framework
• Peer-to-Peer Distributed Object Location and Routing
• Other Aspects
• Conclusion
Analysis Framework (1/2)

• The framework is based on:
  – Identifying the feature space of nonfunctional properties and characteristics of content distribution systems
  – Determining the way in which the nonfunctional properties depend on, and can be affected by, various design features
  – Providing an account, analysis, and evaluation of the design features and solutions that have been adapted by current P2P systems, as well as their shortcomings, potential improvements and proposed alternatives
Analysis Framework (2/2)

Fig. 1. Illustration of the way in which various design features affect the main characteristics of peer-to-peer content distribution systems.
Outline

• Introduction
• Analysis Framework
• Peer-to-Peer Distributed Object Location and Routing
  – Overlay Network Centralization
  – Network Structure
  – Unstructured Architectures
  – Structured Architectures: Freenet, Chord, CAN, Tapestry
• Other Aspects
• Conclusion
P2P Distributed Object Location and Routing

• The operation of any P2P content distribution system relies on a network of peer computers (nodes), and connections (edges) between them.

• “Overlay” network
  – The network is formed on top of the underlying physical computer network.
  – Overlay network can be distinguished in terms of their centralization and structure.
Overlay Network Centralization

• Purely Decentralized Architectures
  – All nodes act both as servers and clients.
  – No central coordination of their activities

• Partially Decentralized Architectures
  – Some of the nodes (supernodes) act as local central indexes for files shared by local peers.
  – No single point of failure: If supernodes fail, the network will automatically replace them with others.

• Hybrid Decentralized Architectures
  – There is a central server facilitating the interaction between peers.
  – “peer-through-peer”, “broker mediated”
  – The central server is a single point of failure.
Network Structure

- **Unstructured**
  - The placement of content is *completely unrelated* to the overlay topology.
  - Searching: brute force methods (flooding in BFS, DFS)
  - More appropriate for accommodating highly-transient node populations
  - E.g., Napster, Publius, Gnutella, Kazaa, Edutella, FreeHaven

- **Structured**
  - The overlay topology is *tightly controlled* and files (or pointers to them) are placed at *precisely specified* locations.
  - A scalable solution for exact-match queries
  - Hard to maintain the structure in the face of a very transient node population
  - E.g., Chord, CAN, PAST, Tapestry

- **Loosely structured**
  - A category of networks that are in between structured and unstructured
  - Although the location of content is *not completely specified*, it is affected by routing hints.
  - E.g., Freenet
## Classification Table

**Table III.** A classification of Peer-to-Peer Content Distribution Systems and Location and Routing Infrastructures in Terms of Their Network Structure, With Some Typical Examples

<table>
<thead>
<tr>
<th>Centralization</th>
<th>Hybrid</th>
<th>Partial</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unstructured</td>
<td>Napster, Publius</td>
<td>Kazaa, Morpheus, Gnutella, Edutella</td>
<td>Gnutella, FreeHaven</td>
</tr>
<tr>
<td>Structured Infrastructures</td>
<td></td>
<td>Chord, CAN, Tapestry, Pastry</td>
<td></td>
</tr>
<tr>
<td>Structured Systems</td>
<td></td>
<td>OceanStore, Mnemosyne, Scan, PAST, Kademlia, Tarzan</td>
<td></td>
</tr>
</tbody>
</table>
Unstructured Architectures (1/3)

- Hybrid Decentralized
  - Central directory server
    - User connection info.
    - File & metadata info.
- Advantages
  - Simple to implement
  - Locate files quickly and efficiently
- Disadvantages
  - Vulnerable to censorship
  - Inherently unscalable
- E.g., Napster, Publius
Unstructured Architectures (2/3)

- Purely Decentralized
  - No central coordination
  - Users (servents) connect to each other directly.
- Gnutella architecture
  - Query: Flooding
    - Send messages to all neighbors
  - Response: Route back
- Scalability Issues
  - With TTL, virtual horizon
  - Without TTL, unlimited flooding
- E.g., Gnutella, FreeHaven
Unstructured Architectures (3/3)

• Partially Centralized
  – Supernodes
    • Indexing & caching files of small subpart of the peer network
    • Peers are automatically elected to become supernodes.

• Advantages
  – Reduced discovery time
  – Normal nodes will be lightly loaded.

• E.g., Kazaa, Edutella, Gnutella (later version)
Structured Network

• The overlay topology is tightly controlled and files (or pointers to them) are placed at precisely specified locations.

• Representative mechanisms
  – Freenet (loosely structured)
  – Chord
  – CAN
  – Tapestry
Freenet (1/4)

• Typical, purely decentralized loosely-structured content distribution system

• Avoiding blindly broadcasting request messages to all neighbors
  – Nodes can produce an estimate of which node is most likely to store certain content.
  – Chain mode propagation
    • Each node makes a local decision about which node to send the request message to next.

• Local data store : Creating a collaborative virtual file system

• Dynamic routing table
  – The addresses of other nodes
  – The files they are thought to hold
Freenet (2/4)

- Files are identified by unique binary keys
  - Based on a hash function (the simplest type)
- Message types
  - Common: Node IDs, hops-to-live, src ID, dst ID
  - Data insert: key, file
  - Data request: key
  - Data reply: file
  - Data failed: failure location (node), reason
- Joining
  - Discover 1 or more existing nodes → Data insert messages
- Inserting a new file
  - First node: Send a data insert message to itself.
  - Receiving node: Check if the key is already taken
    - If taken, the node returns preexisting file as if a request were made for it.
    - If not, the node looks up the closest key (lexicographical distance) in its routing table, and forwards the message to the corresponding node.
Freenet (3/4)

- Searching for a file
  - If a node receives a request for a locally-stored file, the search stops and the data is forwarded back to the requestor.
  - If the node does not store the file, it forwards the request to its neighbor that is *most likely* to have the file, by searching for the file key in its local routing table that is closest to the one requested.
Freenet (4/4)

• Transferring a data and caching
  – If the requested file is eventually found at a certain node, a reply is passed back through each node that forwarded the request to the original node.
  – This data reply message will include the actual data which is cached in all intermediate nodes for future requests.

• Indirect Files
  – To address the problem of obtaining the key
  – When a real file is inserted, the author also inserts a number of indirect files that are named according to search keywords and contain pointers to the real file.
Chord (1/9)

- P2P routing and location infrastructure that performs a **mapping of file identifiers onto node identifiers**.
  - Data location: Identifying data items (*files*) with keys and storing the (*key*, *data item*) pairs at the node that the keys map to.
  - Keys are assigned both to files and nodes by a deterministic function.
Chord (2/9)

- All node identifiers are ordered in an “identifier circle” modulo $2^m$.
- Key $k$ is assigned to the first node whose identifier is equal to, or follows $k$, in the identifier space. ($\text{successor}(k)$)
- When a new node $n$ joins the network, $\text{successor}(n)$’s key is assigned to $n$. 
Chord (3/9)

- **Successor Nodes**

![Diagram of Chord algorithm showing successor nodes and identifier circle.](image-url)

- successor(6) = 0
- successor(2) = 3
- successor(1) = 1

- identifier circle

- identifier
- node
- key
Chord (4/9)

- Node Join and Departure

successor(6) = 7

successor(1) = 3
Chord (5/9)

• Routing & Location
  – The only routing information required is for each node to be aware of its successor node on the circle.
  – Queries for a given key are passed around the circle via these successor pointers.

• Finger table
  – To increase efficiency
    • $O(N) \rightarrow O(\log N)$
  – In this table, each entry $i$ points to the successor of node $n+2^i$. 
Chord (6/9)

- Routing & Location

```
N120
N10
N105
N60
N32
N90
K80
```

"Where is key 80?"

"N90 has K80"
Chord (7/9)

- **Finger Table**

<table>
<thead>
<tr>
<th>start</th>
<th>int.</th>
<th>succ.</th>
<th>keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>0+2^0=1</td>
<td>[1,2)</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>0+2^1=2</td>
<td>[2,4)</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>0+2^2=4</td>
<td>[4,0)</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Node 3 wants to find data 1.
Chord (8/9)

• Join – Finger Table

<table>
<thead>
<tr>
<th>start</th>
<th>int.</th>
<th>succ.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[1,2)</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>[2,4)</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>[4,0)</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>start</th>
<th>int.</th>
<th>succ.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[2,3)</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>[3,5)</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>[5,1)</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>start</th>
<th>int.</th>
<th>succ.</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>[7,0)</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>[0,2)</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>[2,6)</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>start</th>
<th>int.</th>
<th>succ.</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>[4,5)</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>[5,7)</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>[7,3)</td>
<td>0</td>
</tr>
</tbody>
</table>
Chord (9/9)

- **Departure**
  - **Finger Table**

<table>
<thead>
<tr>
<th>start</th>
<th>int.</th>
<th>succ.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[1,2)</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>[2,4)</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>[4,0)</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>start</th>
<th>int.</th>
<th>succ.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[2,3)</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>[3,5)</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>[5,1)</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>start</th>
<th>int.</th>
<th>succ.</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>[7,0)</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>[0,2)</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>[2,6)</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>start</th>
<th>int.</th>
<th>succ.</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>[4,5)</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>[5,7)</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>[7,3)</td>
<td>0</td>
</tr>
</tbody>
</table>
CAN (1/4)

- CAN (Content Addressable Network) is essentially a distributed, Internet-scale hash table that maps file names to their location in the network.
- Each CAN node stores a part ("zone") of the hash table.
- CAN uses a virtual $d$-dimensional Cartesian coordinate space to store ($key \ K$, $value \ V$) pairs.
CAN (2/4)

• Routing & Location
  – Routing follows the straight line. (From A to P: A → B → E)

• Joining
  – New node F chooses random point P and sends a JOIN request to the node covering P.
  – The zone is then split, and half of it is assigned to the new node.
  – New node and neighbors of the split zone update their routing table.
CAN (3/4)

- Routing & Location
CAN (4/4)

• Leaving gracefully
  – Zone & hash table entries are handed over to one of neighbors explicitly.

• Leaving due to failure
  – Detecting failure using periodic update message
  – The neighbor nodes initiate a controlled takeover mechanism.

• Multi-dimensional coordinate space
  – Improving network latency and fault tolerance
Tapestry (1/12)

- Tapestry supports the location of objects and the routing of messages to objects in a distributed, self-administering, and fault-tolerant manner.

- Based on the Plaxton mesh, a distributed data structure.
  - A location & routing mechanism introduced by Plaxton, et al.
  - Each node maintains a neighbor map.

| Table IV. The Neighbor Map Held by Tapestry Node With ID 67493 |
|-----------------------|----------------|----------------|----------------|----------------|----------------|
| Entry 0               | 07493          | x0493          | xx093          | xxx03          | xxxx0          |
| Entry 1               | 17493          | x1493          | xx193          | xxx13          | xxxx1          |
| Entry 2               | 27493          | x2493          | xx293          | xxx23          | xxxx2          |
| Entry 3               | 37493          | x3493          | xx393          | xxx33          | xxxx3          |
| Entry 4               | 47493          | x4493          | xx493          | xxx43          | xxxx4          |
| Entry 5               | 57493          | x5493          | xx593          | xxx53          | xxxx5          |
| Entry 6               | **67493**      | x6493          | xx693          | xxx63          | xxxx6          |
| Entry 7               | 77493          | x7493          | xx793          | xxx73          | xxxx7          |
| Entry 8               | 87493          | x8493          | xx893          | xxx83          | xxxx8          |
| Entry 9               | 97493          | x9493          | xx993          | xxx93          | xxxx9          |

Each entry in this table corresponds to a pointer to another node.
• Messages are incrementally routed to the destination node digit-by-digit, from the right to the left. (suffix matching)
  – cf. prefix matching in JSAC 2004
• E.g., ID 69473 → 34567
  – xxxx7 → xxx67 → xx567 → x4567 → 34567
• Root node for each object
  – To provide a guaranteed node from which the object can be located
  – (object id o, storer id ns) mapping is stored at all nodes from ns to root node.

Fig. 7. Tapestry: Plaxton mesh routing example, showing the path taken by a message originating from node 67493, and destined for node 34567 in a Plaxton mesh, using decimal digits of length 5.
Tapestry (3/12)

• Advantages of Plaxton mesh
  – Simple fault-handling
    • By choosing a node with a similar suffix
  – Scalability
    • With the only bottleneck existing at the root nodes

• Disadvantages of Plaxton mesh
  – The need for global knowledge required for assigning and identifying root nodes
  – The vulnerability of the root nodes
Tapestry (4/12)

- Plaxton mesh assumes static node population.
- Tapestry extends its design to adapt it to the transient populations and provide adaptability, fault tolerance, etc.
  - Back-pointers: used for dynamic node insertion
  - The concept of distance between nodes becomes semantically more flexible. ("closest" node)
  - Maintaining cached content: to recover from failures in routing or object location
  - Multiple roots to each object to avoid single point of failure
  - A set of optimizations improve performance by adapting environment changes.
Tapestry (5/12)
Tapestry (6/12)

Object location
Tapestry (7/12)

• Publishing
  1. Map the object ID to a node ID.
  2. Server (node) sends the message with the object id and server id.
  3. Find the surrogate node as the “root” for the object.
  4. Save the related info there, such as <Object-id, Server-id>.
Tapestry (8/12)

- Surrogate routing
  - Each non-existent ID is mapped to some live node with a similar ID.
  - Looks for a “close” digit in the neighbor maps.
  - Ensuring that arriving at the same unique root node from any location in the Tapestry network.
Tapestry (9/12)

• Location
  1. Client sends the query with the object id.
  2. Route the query to the root node for the object.
  3. Forward the query to the server.

Client: B4F8
Server: B346

Surrogate Routing
Tapestry (10/12)

• Fault-tolerant Routing
  – Detecting link and server failures
    • Relying on TCP timeouts
    • Using back pointers to send periodic heartbeats
  – Operation under faults
    • 2 secondary neighbors
    • Second chance

• Fault-tolerant Location
  – Multiple root nodes
  – Republish location information at regular intervals
Tapestry (11/12)

- Node insertion
  - Need-to-know nodes are notified of N, because N fills a null entry in their routing tables.
  - N might become the new object root for existing objects. References to those objects must be moved to N to maintain object availability.
  - The algorithms must construct a near optimal routing table for N.
  - Nodes near N are notified and may consider using N in their routing tables as an optimization.
• Voluntary node deletion
  – Using backpointers, pointed nodes update their own table with a replacement node.
  – The replacement node republishes.
  – Node N routes references locally rooted objects to their new roots.

• Involuntary node deletion
  – Failure-prone network such as Internet.
  – Periodic beacon to detect outgoing link and node failures.
  – Republishing of object references.
Outline

• Introduction
• Analysis Framework
• Peer-to-Peer Distributed Object Location and Routing
  • Other Aspects
• Conclusion
Other Aspects

- Content Caching, Replication and Migration
- Security
- Provisions for Anonymity
- Provisions for Deniability
- Incentive Mechanisms and Accountability
- Resource Management Capabilities
- Semantic Grouping of Information
Conclusion

• Study of P2P content distribution systems and infrastructures in the view of the analysis framework.

• One of the central characteristics of P2P systems
  – Ability to function, scale, and self-organize in the presence of a highly transient population of nodes and network and computer failures, without the need for a central server administration

• Open research problems
  – The design of new distributed object location, routing and distributed hash table data structure and algorithms
  – The study of more efficient security, anonymity, and censorship resistance schemes
  – The semantic grouping of information in P2P networks
  – The design of incentive mechanisms and reputation systems
  – The convergence of Grid and P2P systems
APPENDIX
CAN: construction

Bootstrap node

new node
1) Discover some node “I” already in CAN
CAN: construction

2) Pick random point in space

(new node)

(x,y)

2) Pick random point in space
CAN: construction

3) I routes to \((x,y)\), discovers node J

new node

3) I routes to \((x,y)\), discovers node J
4) split J’s zone in half... new owns one half
Zone reassignment

Zoning

Partition tree
Zone reassignment

Zoning

Partition tree
Zone reassignment

Zoning

Partition tree
Zone reassignment

Zoning

Partition tree
Tapestry

Dynamic insertion

New NodeID 0x143FE