Outline – A Hierarchical P2P Architecture
and an Efficient Flooding Algorithm

- Introduction
- Hierarchical Architecture
- Backbone Routing
- Experiments
- Conclusion
Outline

- Introduction
- Hierarchical Architecture
- Backbone Routing
- Experiments
- Conclusion

What is P2P

- Every node provides and gets some service
- Each node potentially has the same responsibility
- Sharing can be in different ways:
  - CPU cycles: SETI@Home
  - Storage space: Napster, Gnutella, Freenet...
Why P2P

- Cost sharing
- Resource aggregation
- Improved scalability/reliability
- Increased autonomy
- Anonymity/privacy
- Dynamism
- Ad-hoc communication

Related work

- Three categories:
  - Centralized
    - Napster
  - Decentralized but structured
    - Tapestry, Pastry, Chord, CAN
  - Decentralized and unstructured
    - Gnutella
Motivation: P2P not perfect

- Structured systems
  - Not applicable to typical Internet Environment
  - High requirement on topology
  - Search by identifier

- Unstructured systems
  - Lack of scalability
  - Inefficient search mechanism

Proposed approach

- Build a cluster based hierarchical model and a backbone routing algorithm to increase scalability, improve search mechanism, and provide load balancing under the unstructured scenario
Outline

- Introduction
- Hierarchical Architecture
- Backbone Routing
- Experiments
- Conclusion

Hierarchical structure vs. Gnutella structure
Hierarchical structure – continued ..

System Architecture

Main components(1):
Well-known register server

Well-known register server structure
Main components(2)
super-peer

Super-Peer Structure

Main components(3)
client peer

Client Peer Structure
Main components(4) backup peer

- super-peer: a potential one-point failure
- Backup peer: selected from client peers, periodically backup the index library of the super-peer
- possibility of both the super-peer and backup peer failing at the same time is small

Outline

- Introduction
- Hierarchical Architecture
- Backbone Routing
- Experiments
- Conclusion
Backbone routing - motivation

- unstructured systems: no clue for where content is placed, queries have to be flooded through the network to get results

Backbone routing – basic idea

- Control flooding:
  - several different paths may exist to connect two nodes
  - If node v can anticipate that one of its neighbors u, receives query messages from another path, then v does not forward the query to u.
  - Make a rule directing the nodes forwarding messages.
  - Keep track of 2-hop topology information to compute the forwarding set.
Backbone routing – algorithm (1)

Definitions used in routing algorithm

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>Current node</td>
</tr>
<tr>
<td>id(v)</td>
<td>Node v’s unique id</td>
</tr>
<tr>
<td>n(v)</td>
<td>Neighbor set of v</td>
</tr>
<tr>
<td>fr(u,v)</td>
<td>forward reaching set of u for the current node v</td>
</tr>
<tr>
<td>routing(u,v)</td>
<td>for local source u, current node v’s routing set</td>
</tr>
</tbody>
</table>

Backbone routing – algorithm (2)

Algorithm to compute routing table:

\[
fr(u,v) = N(u) \cup \{ \text{all } v' \text{ in } N(r) \mid r \in N(u) \text{ And } \text{id}(r) < \text{id}(v) \}
\]

where \( N(u) = N(u,v) = n(u) \) excluding v

\[
routing(u,v) = \text{all } v' \text{ in } N(v), \text{ such that}
\]

1. \( v' \notin fr(u,v) \text{ AND} \)
2. \( \{N(v') \cap fr(u,v) = \emptyset \} \text{ OR} \)
   \( \{N(v') \cap fr(u,v) = A, \forall v'' \in A, \text{id}(v'') > \text{id}(v)\} \)
Backbone routing – algorithm (3)

`forward(u,v)`

/* when node v receives forwarded query from its neighbor u, this algorithm decides how v forwards this query */

If (query has been received before)
    discard it
else
    if (u is null) /* v is node which initiates query*/
        forward the query to N(v)
    else
        forward the query to routing(u,v)

Backbone routing – an example

N5 receives forwarded message from N1, the routing algorithm decide which neighbors N5 will forward the message to.
Backbone routing – an example

In flooding, N5 will forward the query to all other neighbors, except N1.

To avoid duplication, if N5 knows some of its neighbors may be covered by \( \text{fr}(N1) \), then N5 needs not forward the message to those nodes.

So N5 need to know N1’s forward reaching set \( \text{fr}(N1,N5) \).
Backbone routing – an example

fr(N1,N5) includes N1’s neighbors: N2,N7
And ...

fr(N1,N5) includes N1’s neighbors: N2,N7
It also include N2’s neighbors, since id(N2)<id(N5)
So fr(N1,N5)={N2,N7,N3,N4}
Backbone routing – an example

So we can decide route(N1,N5) now. 3 neighbor candidates: N3,N6,N8

N3 \in \text{fr}(N1,N5), so N5 don't forward to N3.

N6's neighbor N4 \in \text{fr}(N1,N5), and id(N4)<id(N5), so N5 don't forward to it, only N8 left.

1. N5 routing table via Flooding

<table>
<thead>
<tr>
<th>Neighbor u</th>
<th>routing (u, N5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>N3, N6, N8</td>
</tr>
<tr>
<td>N3</td>
<td>N1, N6, N8</td>
</tr>
<tr>
<td>N6</td>
<td>N1, N3, N8</td>
</tr>
<tr>
<td>N8</td>
<td>N1, N3, N6</td>
</tr>
</tbody>
</table>

2. N5 routing table via Efa

<table>
<thead>
<tr>
<th>Neighbor u</th>
<th>routing (u, N5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>N8</td>
</tr>
<tr>
<td>N3</td>
<td>N8</td>
</tr>
<tr>
<td>N6</td>
<td></td>
</tr>
<tr>
<td>N8</td>
<td>N1, N3</td>
</tr>
</tbody>
</table>
Peer Instruction: Backbone routing

<table>
<thead>
<tr>
<th>Neighbor u</th>
<th>routing (u, N8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>N5</td>
<td>?</td>
</tr>
<tr>
<td>N6</td>
<td>?</td>
</tr>
<tr>
<td>N7</td>
<td>?</td>
</tr>
<tr>
<td>N9</td>
<td>?</td>
</tr>
</tbody>
</table>

N8 routing table via Efa

Backbone routing – an example

<table>
<thead>
<tr>
<th>Neighbor u</th>
<th>routing (u, N8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>N5</td>
<td>?</td>
</tr>
<tr>
<td>N6</td>
<td>?</td>
</tr>
<tr>
<td>N7</td>
<td>?</td>
</tr>
<tr>
<td>N9</td>
<td>?</td>
</tr>
</tbody>
</table>

N8 routing table via Efa

\[ \text{fr}(N5, N8) = \{N1, N3, N6, N2, N7, N4\} \]
\[ \text{fr}(N6, N8) = \{N4, N5, N2, N3, N1\} \]
\[ \text{fr}(N7, N8) = \{N1, N9, N2, N5\} \]
\[ \text{fr}(N9, N8) = \{N4, N7, N2, N3, N1\} \]
Backbone routing – an example

\[ \text{fr}(N5,N8) = \{N1,N3,N6, N2,N7, N4\} \text{ e.g. } N6,N7 \notin \text{fr}(N5,N8), \]
\[ \text{fr}(N6,N8) = \{N4,N5, N2,N3, N1\} \text{ | N9 does not sat (2)} \]
\[ \text{fr}(N7,N8) = \{N1,N9, N2,N5\} \]
\[ \text{fr}(N9,N8) = \{N4,N7, N2,N3, N1\} \]

N8 routing table via Efa

<table>
<thead>
<tr>
<th>Neighbor u</th>
<th>routing (u, N8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>N5</td>
<td>Nil (Drop)</td>
</tr>
<tr>
<td>N6</td>
<td>Nil (Drop)</td>
</tr>
<tr>
<td>N7</td>
<td>N6</td>
</tr>
<tr>
<td>N9</td>
<td>Nil (Drop)</td>
</tr>
</tbody>
</table>

Backbone Routing – Protocol

- Data structure
- Join
- Probe
- Update
- Query
Outline

- Introduction
- Hierarchical Architecture
- Backbone Routing
- Experiments
- Conclusion

Routing protocol simulation

- **Simulator:**
  - event-based, written in Java
  - simulate the application level routing
  - topology: BarabasiAlbert, random, grid
System overhead vs. network size

Routing protocol simulation - continued

Degree vs. duplication  
TTL vs. system overhead
Real system evaluation

- Experiment setup
  - 16 PCs
  - Intel Pentium III 1GHz processor
  - 256M of RAM
  - Red Hat Linux 9
  - 50 different files
  - every peer maintains 20 files
  - each file around 5KB.

Real system evaluation - continued

- Success rate vs. system overhead
- Query hits vs. completion time
Outline

- Introduction
- Hierarchical Architecture
- Backbone Routing
- Experiments
- Conclusion

Conclusion

- Investigate the current P2P systems
- An Efficient Model
  - Hierarchical architecture
  - Efficient backbone routing
- Prototype implementation and evaluation