Peer-to-Peer Systems

DHT examples, part 2
(Pastry, Tapestry and Kademlia)

Michael Welzl  michael.welzl@uibk.ac.at
DPS NSG Team  http://dps.uibk.ac.at
Institute of Computer Science
University of Innsbruck, Austria

Plaxton routing

- Plaxton, Rajamaran and Richa: mechanism for efficient dissemination of objects in a network, published in 1997
  - Before P2P systems came about!

- Basic idea: prefix-oriented routing (fixed number of nodes assumed)
  - Object with ID A is stored at the node whose ID has the longest common prefix with A
    - If multiple such nodes exist, node with longest common suffix is chosen
  - Goal: uniform data dissemination
  - Routing based on pointer list (object - node mapping) and neighbor list (primary + secondary neighbors)
  - Generalization of routing on a hypercube

- Basis for well known DHTs Pastry, Tapestry (and follow-up projects)
  - Method adapted to needs of P2P systems + simplified
Pastry: Topology

- Identifier space:
  - $2^l$-bit identifiers (typically: $l = 128$), wrap-around at $2^l - 1 \equiv 0$
  - interpret identifiers to the base of $2^b$ (typically: $b = 4$, base 16)
  - prefix-based tree topology
  - leaves can be keys and node IDs
  - (key, value) pairs managed by numerically closest node

Pastry: Routing Basics

- Goal: find node responsible for $k$, e.g. 120
- Tree-based search for lookup($k$)
  - Traverse tree search structure top-down
- Prefix-based routing for lookup($k$)
  - Approximate tree search in distributed scenario
  - Forward query to known node with longest prefix matching $k$
**Pastry: Routing Basics /2**

- Routing in Pastry:
  - In each routing step, query is routed towards "numerically" closest node
  - That is, query is routed to a node with a one character longer prefix (= \( b \) Bits)

\[ O(\log_2 N) \] routing steps

- If that is not possible:
  - route towards node that is numerically closer to ID

---

**Pastry: Routing Basics /3**

- Example:
  - Node-ID = 0221
  - Base = 3 (not power of 2, because it is easier to draw :-)

---

*Images and diagrams related to the text.*
Pastry: Routing Basics /4

- Data (key-value-pairs) are managed in numerically closest node
  - keys → nodes:
    0002 → 0002, 01** → 0110

- Linking between Prefix-areas:
  - Nodes within a certain prefix area know IP addresses of each other
  - Each node in a prefix area knows one or more nodes from another prefix area

- From which prefix areas should a node know other nodes?
  - Links to shorter-prefix node areas on each prefix level

Pastry: Routing Basics /5

- Example:
  - Node in area 222* knows nodes from prefix areas 220*, 221* & 20**, 21** & 0***, 1***
  - Logarithmic number of links:
    - For prefix-length p: (base-1) links to other nodes with prefix length p, but with a different digit at position p
    - \( \log(N) \) different prefix-lengths: 1 – log(N)
Pastry: Routing Information

- **Challenges**
  - Efficiently distribute search tree among nodes
  - Honor network proximity

- **Pastry routing data per node**
  - **Routing table**
    - Long-distance links to other nodes
  - **Leaf set**
    - Numerically close nodes
  - **Neighborhood set**
    - Close nodes based on proximity metric (typically ping latency)

Pastry: Routing Table

- **Routing table**
  - Long distance links to other prefix realms
  - l/b rows: one per prefix length
  - $2^b-1$ columns: one per digit different from local node ID
  - Routing table for node 120:
Pastry: Routing Table

- \( \left[ \log_b N \right] \) rows with \( 2^{b-1} \) entries each
  - row \( i \): hold IDs of nodes whose ID share an \( i \)-digit prefix with node
  - column \( j \): digit(\( i+1 \)) = \( j \)
  - Contains topologically closest node that meets these criteria
- Example: \( b=2 \), Node-ID = 32101

Pastry: Routing Information

- Leaf set
  - contains numerically closest nodes (\( l/2 \) smaller and \( l/2 \) larger keys)
  - fixed maximum size
  - similar to Chord’s succ/pred list
  - for routing and recovery from node departures
- Neighbor set
  - contains nearby nodes
  - fixed maximum size
  - scalar proximity metric assumed to be available
    - e.g., IP hops, latency
    - irrelevant for routing
    - ‘cache’ of nearby candidates for routing table
**Pastry Routing Algorithm**

- **Routing of packet with destination K at node N:**
  1. Is K in Leaf Set, route packet directly to that node
  2. If not, determine common prefix (N, K)
  3. Search entry T in routing table with prefix (T, K) > prefix (N, K), and route packet to T
  4. If not possible, search node T with longest prefix (T, K) out of merged set of routing table, leaf set, and neighborhood set and route to T
    - This was shown to be a rare case
  - Access to routing table $O(1)$, since row and column are known
  - Entry might be empty if corresponding node is unknown

**Pastry: Routing Procedure**

- **Long-range routing**
  - if key $k$ not covered by leaf set:
  - forward query for $k$ to
    - node with longer prefix match than self or
    - same prefix length but numerically closer

![Node Diagram](image)
Pastry: Routing Procedure

- **Close-range routing**
  - $k$ covered by nodes IDs in leaf set
  - pick leaf node $n_i$ numerically closest to $k$
  - $n_i$ must be responsible for $k$ — last step in routing procedure
  - return $n_i$ as answer to query for $k$

Another example

- **Key = 01200**
  - Common prefix: 32101
  - 01200
  - Common prefix: 3201

- **Key = 32200**
  - Common prefix: 32101
  - 32200
  - Common prefix: 32101

- **Key = 33122**
  - Common prefix: 32101
  - 33122
  - Common prefix: 32101

Node-ID = 32101

```
Routing table
```

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>Key</th>
<th>Common prefix</th>
<th>Leaf set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>01234</td>
<td>01200</td>
<td>32101</td>
<td>32010</td>
</tr>
<tr>
<td>1</td>
<td>3211</td>
<td>32130</td>
<td>32101</td>
<td>32010</td>
</tr>
<tr>
<td>2</td>
<td>32012</td>
<td>32121</td>
<td>32101</td>
<td>32121</td>
</tr>
<tr>
<td>3</td>
<td>32100</td>
<td>32103</td>
<td>32101</td>
<td>32103</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Node-ID = 32302

```
Routing table
```

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>Key</th>
<th>Common prefix</th>
<th>Leaf set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>01234</td>
<td>01200</td>
<td>32101</td>
<td>32010</td>
</tr>
<tr>
<td>1</td>
<td>3211</td>
<td>32130</td>
<td>32101</td>
<td>32010</td>
</tr>
<tr>
<td>2</td>
<td>32012</td>
<td>32121</td>
<td>32101</td>
<td>32121</td>
</tr>
<tr>
<td>3</td>
<td>32100</td>
<td>32103</td>
<td>32101</td>
<td>32103</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Node-ID = 32120

```
Routing table
```

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>Key</th>
<th>Common prefix</th>
<th>Leaf set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>01234</td>
<td>01200</td>
<td>32101</td>
<td>32010</td>
</tr>
<tr>
<td>1</td>
<td>3211</td>
<td>32130</td>
<td>32101</td>
<td>32010</td>
</tr>
<tr>
<td>2</td>
<td>32012</td>
<td>32121</td>
<td>32101</td>
<td>32121</td>
</tr>
<tr>
<td>3</td>
<td>32100</td>
<td>32103</td>
<td>32101</td>
<td>32103</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Node-ID = 32123

```
Routing table
```

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>Key</th>
<th>Common prefix</th>
<th>Leaf set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>01234</td>
<td>01200</td>
<td>32101</td>
<td>32010</td>
</tr>
<tr>
<td>1</td>
<td>3211</td>
<td>32130</td>
<td>32101</td>
<td>32010</td>
</tr>
<tr>
<td>2</td>
<td>32012</td>
<td>32121</td>
<td>32101</td>
<td>32121</td>
</tr>
<tr>
<td>3</td>
<td>32100</td>
<td>32103</td>
<td>32101</td>
<td>32103</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Node-ID = 32102

```
Routing table
```

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>Key</th>
<th>Common prefix</th>
<th>Leaf set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>01234</td>
<td>01200</td>
<td>32101</td>
<td>32010</td>
</tr>
<tr>
<td>1</td>
<td>3211</td>
<td>32130</td>
<td>32101</td>
<td>32010</td>
</tr>
<tr>
<td>2</td>
<td>32012</td>
<td>32121</td>
<td>32101</td>
<td>32121</td>
</tr>
<tr>
<td>3</td>
<td>32100</td>
<td>32103</td>
<td>32101</td>
<td>32103</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Node-ID = 32121

```
Routing table
```

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>Key</th>
<th>Common prefix</th>
<th>Leaf set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>01234</td>
<td>01200</td>
<td>32101</td>
<td>32010</td>
</tr>
<tr>
<td>1</td>
<td>3211</td>
<td>32130</td>
<td>32101</td>
<td>32010</td>
</tr>
<tr>
<td>2</td>
<td>32012</td>
<td>32121</td>
<td>32101</td>
<td>32121</td>
</tr>
<tr>
<td>3</td>
<td>32100</td>
<td>32103</td>
<td>32101</td>
<td>32103</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Node-ID = 32110

```
Routing table
```

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>Key</th>
<th>Common prefix</th>
<th>Leaf set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>01234</td>
<td>01200</td>
<td>32101</td>
<td>32010</td>
</tr>
<tr>
<td>1</td>
<td>3211</td>
<td>32130</td>
<td>32101</td>
<td>32010</td>
</tr>
<tr>
<td>2</td>
<td>32012</td>
<td>32121</td>
<td>32101</td>
<td>32121</td>
</tr>
<tr>
<td>3</td>
<td>32100</td>
<td>32103</td>
<td>32101</td>
<td>32103</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Arrival of a new node

- Node X wants to join Pastry DHT
  - Determine NodeID of X → 12333 (hash of IP address)
  - Initialize tables at node X
  - Send JOIN message to key 12333 via topologically nearest Pastry node
  - Node currently in charge of this key: Z

### Diagram 1

- Node X joins the network with NodeID 12333.
- Node X initializes tables and sends a JOIN message to key 12333.
- The node currently in charge of this key is Z.

### Table 1

<table>
<thead>
<tr>
<th>i \ j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Neighbor Set

- Node X copies Neighbor-Set from node A0.

### Diagram 2

- Node X joins the network with NodeID 12333.
- Node X initializes tables and sends a JOIN message to key 12333.
- The node currently in charge of this key is Z.

### Table 2

<table>
<thead>
<tr>
<th>i \ j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Neighbor Set

- Node X copies Neighbor-Set from node A0.
Arrival of a new node /3

- Node X wants to join Pastry DHT
  - Node A0 routes message to node Z
  - Each node sends row in routing table to X
  - Here A0

<table>
<thead>
<tr>
<th>i</th>
<th>\ j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>02231</td>
<td>13231</td>
<td>32331</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

< Node-ID | > Node-ID

Arrival of a new node /4

- Node X wants to join Pastry DHT
  - Node A0 routes message to node Z
  - Each node sends row in routing table to X
  - Here A1

<table>
<thead>
<tr>
<th>i</th>
<th>\ j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>02231</td>
<td>13231</td>
<td>32331</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10122</td>
<td>11312</td>
<td>12222</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

< Node-ID | > Node-ID
Arrival of a new node /5

- Node X wants to join Pastry DHT
  - Node A0 routes message to node Z
  - Each node sends row in routing table to X
  - Here A2

<table>
<thead>
<tr>
<th>i</th>
<th>\ j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>02231</td>
<td>13231</td>
<td>32331</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10122</td>
<td>11312</td>
<td>12222</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>12033</td>
<td>12111</td>
<td>12311</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(<\text{Node-ID}\) \quad \(>\text{Node-ID}\)

\(X = 12333\)

\(A_0 = 23231\)
\(A_1 = 13231\)
\(A_2 = 12222\)
\(A_3 = 12311\)

Arrival of a new node /6

- Node X wants to join Pastry DHT
  - Node A0 routes message to node Z
  - Each node sends row in routing table to X
  - Here A3

<table>
<thead>
<tr>
<th>i</th>
<th>\ j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>02231</td>
<td>13231</td>
<td>32331</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10122</td>
<td>11312</td>
<td>12222</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>12033</td>
<td>12111</td>
<td>12311</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>12301</td>
<td>12320</td>
<td>12332</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(<\text{Node-ID}\) \quad \(>\text{Node-ID}\)

\(X = 12333\)

\(A_0 = 23231\)
\(A_1 = 13231\)
\(A_2 = 12222\)
\(A_3 = 12311\)
\(A_4 = Z = 12332\)
Arrival of a new node /7

- Node X wants to join Pastry DHT
  - Node A0 routes message to node Z
  - Each node sends row in routing table to X
  - Here A4

Arrival of a new node /8

- Node X wants to join Pastry DHT
  - Node Z copies its Leaf-Set to Node X
Arrival of a new node /9

- Some entries are doubtable
  - Entries pointing to "own-ID-positions" not required

- Some are missing
  - Take the node-IDs just visited

\[
\begin{array}{cccc}
| i \setminus j | 0 & 1 & 2 & 3 \\
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>
\end{array}
\]

\[
\begin{array}{cccc}
<table>
<thead>
<tr>
<th>&lt; \text{Node-ID}</th>
<th>&gt; \text{Node-ID}</th>
</tr>
</thead>
<tbody>
<tr>
<td>12331</td>
<td>12322</td>
</tr>
<tr>
<td>12331</td>
<td>12330</td>
</tr>
</tbody>
</table>
\end{array}
\]

Arrival of a new node /10

- Node X wants to join
  Pastry DHT
  - Node x sends its routing table to each neighbor

\[
\begin{array}{cccc}
| i \setminus j | 0 & 1 & 2 & 3 \\
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>
\end{array}
\]

\[
\begin{array}{cccc}
<table>
<thead>
<tr>
<th>&lt; \text{Node-ID}</th>
<th>&gt; \text{Node-ID}</th>
</tr>
</thead>
<tbody>
<tr>
<td>12331</td>
<td>12322</td>
</tr>
<tr>
<td>12331</td>
<td>12330</td>
</tr>
</tbody>
</table>
\end{array}
\]
Arrival of a new node /11

- Efficiency of initialization procedure
  - Quality of routing table \((b=4, l=16, 5k \text{ nodes})\)

SL: transfer only the \(i\)th routing table row of \(A_i\)

WT: transfer of \(i\)th routing table row of \(A_i\) as well as analysis of leaf and neighbor set

WTF: same as WT, but also query the newly discovered nodes from WT and analyse data

Failure of Pastry Nodes

- Detection of failure
  - Periodic verification of nodes in Leaf Set
    - "Are you alive" also checks capability of neighbor
  - Route query fails

- Replacement of corrupted entries
  - Leaf-Set
    - Choose alternative node from Leaf \((L) \cup \text{Leaf (±|L|/2)}\)
    - Ask these nodes for their Leaf Sets
  - Entry \(R_{x,y}\) in routing table failed:
    - Ask neighbor node \(R_{x,i}\) of same row for route to \(R_{x,y}\)
    - If not successful, test entry \(R_{x,i+1}\) in next row
Performance Evaluation

- Routing Performance
  - Number of Pastry hops (b=4, l=16, $2 \cdot 10^5$ queries)
  - $O(\log N)$ for number of hops in the overlay

- Overhead of overlay (in comparison to route between two node in the IP network)
  - But: Routing table has only $O(\log N)$ entries instead of $O(N)$

Locality

- In routing, if multiple peers match, the next hop is chosen based on some metric
  - Typically RTT

- This is done based on local information
  - May not generally route in the right direction

- Expected latency grows with every hop
  - Last hops most expensive; but: the closer we get to the destination, the more likely it is that the leaf set can be used
Summary Pastry

- Complexity:
  - $O(\log N)$ hops to destination
  - Often even better through Leaf- and Neighbor-Set: $O(\log_2 N)$
  - $O(\log N)$ storage overhead per node

- Good support of locality
  - Explicit search of close nodes (following some metric)

- Used in many applications
  - PAST (file system), Squirrel (Web-Cache), ...
  - Many publications available, open source implementation: FreePastry

Tapestry

- Tapestry developed at UC Berkeley
  - Different group from CAN developers

- Tapestry developed in 2000, but published in 2004
  - Originally only as technical report, 2004 as journal article

- Many follow-up projects on Tapestry
  - Example: OceanStore

- Like Pastry, based on work by Plaxton et al.

- Pastry was developed at Microsoft Research and Rice University
  - Difference between Pastry and Tapestry minimal
  - Tapestry and Pastry add dynamics and fault tolerance to Plaxton network
Tapestry: Routing Mesh

- (Partial) routing mesh for a single node 4227
  - Neighbors on higher levels match more digits

```

```

Tapestry: Neighbor Map for 4227

<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>8</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1D76</td>
<td>27AB</td>
<td></td>
<td>51E5</td>
<td>6F43</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>43C9</td>
<td>44AF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>43C9</td>
<td>44AF</td>
<td></td>
<td></td>
<td></td>
<td>42A2</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4228</td>
</tr>
</tbody>
</table>
```

- There are actually 16 columns in the map (base 16)
- Normally more entries would be filled (limited by a constant)
- Tapestry has multiple neighbor maps
Tapestry: Routing Example

- Route message from 5230 to 42AD
- Always route to node closer to target
  - At nth hop, look at n+1 level in neighbor map → "always" one digit more
- Not all nodes and links are shown

Tapestry: Properties

- Node responsible for objects which have the same ID
  - Unlikely to find such node for every object
  - Node also responsible for "nearby" objects (surrogate routing, see below)

- Object publishing
  - Responsible nodes only store pointers
  - Multiple copies of object possible
  - Each copy must publish itself
  - Pointers cached along the publish path
  - Queries routed towards responsible node
  - Queries "often" hit cached pointers
  - Queries for same object go (soon) to same nodes

- Note: Tapestry focuses on storing objects
  - Chord and CAN focus on values, but in practice no difference
**Tapestry: Publishing Example**

- Two copies of object "DOC" with ID 4377 created at AA93 and 4228
- AA93 and 4228 publish object DOC, messages routed to 4377
  - Publish messages create location pointers on the way
- Any subsequent query can use location pointers

**Tapestry: Querying Example**

- Requests initially route towards 4377
- When they encounter the publish path, use location pointers to find object
- Often, no need to go to responsible node
- Downside: Must keep location pointers up-to-date
Tapestry: Making It Work

- Previous examples show a Plaxton network
  - Requires global knowledge at creation time
  - No fault tolerance, no dynamics

- Tapestry adds fault tolerance and dynamics
  - Nodes join and leave the network
  - Nodes may crash
  - Global knowledge is impossible to achieve

- Tapestry picks closest nodes for neighbor table
  - Closest in IP network sense (= shortest RTT)
  - Network distance (usually) transitive
    - If A is close to B, then B is also close to A
  - Idea: Gives best performance

Tapestry: Fault-Tolerant Routing

- Tapestry keeps mesh connected with keep-alives
  - Both TCP timeouts and UDP "hello" messages
  - Requires extra state information at each node

- Neighbor table has backup neighbors
  - For each entry, Tapestry keeps 2 backup neighbors
  - If primary fails, use secondary
    - Works well for uncorrelated failures

- When node notices a failed node, it marks it as invalid
  - Most link/connection failures short-lived
  - Second chance period (e.g., day) during which failed node can come back and old route is valid again
  - If node does not come back, one backup neighbor is promoted and a new backup is chosen
Tapestry: Fault-Tolerant Location

- Responsible node is a single point of failure

- **Solution:** Assign multiple roots per object
  - Add "salt" to object name and hash as usual
  - Salt = globally constant sequence of values (e.g., 1, 2, 3, ...)

- Same idea as CAN's multiple realities

- This process makes data more available, even if the network is partitioned
  - With $s$ roots, availability is $P = 1 - (1/2)^s$
  - Depends on partition

- These two mechanisms "guarantee" fault-tolerance
  - In most cases :-)
  - Problem: If the only out-going link fails...

---

Tapestry: Surrogate Routing

- Responsible node is node with same ID as object
  - Such a node is unlikely to exist

- **Solution:** surrogate routing

- What happens when there is no matching entry in neighbor map for forwarding a message?
  - Node (deterministically) picks next entry in neighbor map
    - If that one also doesn’t exist, next of next ... and so on

- **Idea:** If "missing links" are deterministically picked, any message for that ID will end up at same node
  - This node is the surrogate

- If new nodes join, surrogate may change
  - New node is neighbor of surrogate
**Surrogate Routing Example**

Peer 2716 searches for 4666:

Level 1, current digit j = 4

Level 2, j = 6 doesn't exist, next link: j = 8

Level 3, j = 6

Node 4860 doesn't have any level 4 neighbors => done

---

**Tapestry: Performance**

- Messages routed in $O(\log_b N)$ hops
  - At each step, we resolve one more digit in ID
  - $N$ is the size of the namespace (e.g., SHA-1 = 40 digits)
  - Surrogate routing adds a bit to this, but not significantly

- State required at a node is $O(b \log_b N)$
  - Tapestry has $c$ backup links per neighbor, $O(c b \log_b N)$
  - Additionally, same number of backpointers
Complexity comparison of DHTs so far

<table>
<thead>
<tr>
<th></th>
<th>CAN</th>
<th>Chord</th>
<th>Pastry</th>
<th>Tapestry</th>
</tr>
</thead>
<tbody>
<tr>
<td>States per node</td>
<td>$O(D)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
</tr>
<tr>
<td>Pathlength (Routing)</td>
<td>$O\left(\frac{D}{N} N^{\frac{1}{2}}\right)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
</tr>
<tr>
<td>Join of node</td>
<td>$O(DN^{\frac{1}{2}})$</td>
<td>$O(\log^2 N)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
</tr>
<tr>
<td>Leave of node</td>
<td>?</td>
<td>$O(\log^2 N)$</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

Kademlia

- From New York University
  - Used in eMule, Overnet, Azureus, ...

- Overlay:
  - Tree
  - Node Position:
    - shortest unique prefix
  - Service:
    - Locate closest nodes to a desired ID

- Routing:
  - "based on XOR metric"
  - keep $k$ nodes for each sub-tree which shares the root as the sub-trees where $p$ resides.
    - Share the prefix with $p$
    - Magnitude of distance (XOR)
    - $k$: replication parameter (e.g. 20)
Kademlia - Hashing and distance

- Routing idea similar to Plaxton’s mesh: improve closeness one bit at a time
- Nodes and Keys are mapped to m-bit binary strings
- Distance between two identifiers: the XOR string, as a binary number

\[
\begin{align*}
x &= 0 1 0 1 1 0 \\
y &= 0 1 1 0 1 1 \\
x \oplus y &= 0 0 1 1 0 1 \\
d(x, y) &= 13
\end{align*}
\]

- If \( x \) and \( y \) agree in the first \( i \) digits and disagree in the \((i+1)\)th digit, then \( 2^i \leq d(x, y) \leq 2^{i+1} - 1 \)

\[
\begin{align*}
x &= 0 1 0 1 1 0 & x &= 0 1 0 1 1 0 \\
y &= 0 1 1 1 1 0 & y &= 0 1 1 0 0 1 \\
x \oplus y &= 0 0 1 0 0 0 & x \oplus y &= 0 0 1 1 1 1 \\
d(x, y) &= 8 & d(x, y) &= 15
\end{align*}
\]

Kademlia - Routing table

- Each node with ID \( x \) stores \( m \) \( k \)-buckets
  - a \( k \)-bucket stores \( k \) nodes that are at distance \([2^i, 2^{i+1} - 1]\)
    - empty bucket if no nodes are known
    - Continuous simple queries for values in \( k \)-buckets are used to refresh \( k \)-buckets
      - full \( k \)-bucket: least-recently used node is removed
- Tables are updated when lookups are performed
- Due to XOR symmetry a node receives lookups from the nodes that are in its own table
- Node Joins
  - contact a participating node and insert it in the appropriate bucket
  - perform a query for your own ID
  - refresh all buckets
Kademlia - Lookups

- Process is iterative:
  - everything is controlled by the initiator node
  - query in parallel the $\alpha$ nodes closest to the query ID
    - Parallel search: fast lookup at the expense of increased traffic
    - nodes return the $k$ nodes closest to the query ID
    - go back to step 1, and select the $\alpha$ nodes from the new set of nodes
    - Terminate when you have the $k$ closest nodes

- Key lookups are done in a similar fashion, but terminate when key is found
  - the requesting node cashes the key locally

- Underlying invariant:
  - If there exists some node with ID within a specific range then $k$-bucket is not empty
  - If the invariant is true, then the time is logarithmic
  - we move one bit closer each time
  - Due to refreshes the invariant holds with high probability

Kademlia vs. Chord and Pastry

- Comparing with Chord
  - Like Chord: achieves similar performance
    - deterministic
    - $O(\log N)$ contacts (routing table size)
    - $O(\log N)$ steps for lookup service (?)
    - Lower node join/leave cost
  - Unlike Chord:
    - Routing table: view of the network
    - Flexible Routing Table
    - Given a topology, there are more than one routing table
    - Symmetric routing

- Comparing with Pastry
  - Both have flexible routing table
  - Better analysis properties
References / acknowledgments

- Slides from:
  - Jussi Kangasharju
  - Christian Schindelhauer
  - Klaus Wehrle