Frequently asked questions from the previous class survey

- Pastry: Simplified routing:
  - How do you choose directions initially?
  - Are hops large when the ID space is sparse?
  - What should be the size of the leaf set?
  - How do nodes detect failures in the leaf set? Heartbeats?
- Pastry: Core routing
  - "n" in the routing table: It's a placeholder...
  - Can two cells have the same destination address?
  - How does Pastry find a nearest (network) neighbor?
  - When a new node is added and all nodes are sending their information, is privacy compromised?
- Meshnets

Topics covered in this lecture

- Tapestry
- Chord

Tapestry

- Routes messages to nodes based on GUIDs associated with the resources
- Uses prefix routing in a manner similar to Pastry
- 160-bit identifiers are used
  - To refer to both objects and nodes that perform routing actions
  - For any resource with GUID G, there is a unique root node, with GUID R_G
    - R_G is numerically closest to G

Tapestry Routing [Summary]

- Uses local routing tables, which they also call neighbor maps, to route messages
- Routing is digit-by-digit
  - 4*** → 42** → 42A* → 42AD
- This longest prefix routing is also used by classless interdomain routing (CIDR)
Tapestry: Routing messages

- Each node maintains a routing table
- Entries include node IDs and IP addresses
- This routing table has multiple levels
  - Each level contains links to nodes matching a prefix up to a digit position in the ID
  - The \(i\)th entry in the \(j\)th level at node \(N\)
    - Location of the closest node which begins with the prefix \((N, j-1) + i\)
    - E.g. 9th entry of the 4th level for node 325AE is 3259

When a digit cannot be matched?

- Looks for a “close” digit in the routing table
- This approach is called surrogate routing
- Results in mapping every identifier \(G\) to a unique root node \(G_R\)

Managing a dynamic environment

- Route reliably even when intermediate links are changing or faulty
- Exploit network path diversity
  - Via redundant routing paths
- Primary links are augmented by backup-links
  - Each sharing the same prefix

Managing multiple copies of the resource

- Hosts \(H\) holding replicas of \(G\) periodically invoke \(\text{publish}(G)\)
  - Ensures that newly arrived hosts become aware of the existence of \(G\)
- On each invocation of \(\text{publish}(G)\)
  - Message is routed from invoker towards node \(R_G\)
  - On receipt of a publish message \(R_G\) enters \((G, \text{IP}_G)\)
    - The mapping between \(G\) and IP address of \(H\)
  - Each node in the publication path caches the same mapping

Managing multiple copies of the resource

- When nodes hold multiple \((G, \text{IP})\) mappings for the same GUID?
  - They are sorted by network distance to the IP address
- Results in selection of nearest available replica of the object
An example of managing replicas using Tapestry

The Chord System
- Assigns IDs to keys and nodes from the same 1-dimensional ID space
- Nodes are organized into a ring
- Data item with key $k$ is mapped to a node with the smallest $id \geq k$
  - Also referred to as $\text{successor}(k)$

Chord lookups
- $N$ is the number of possible nodes in the system
- Each node maintains a **finger table**
  - With $\log N$ entries
  - Entries contain IP addresses of nodes
    - Half-way around the ID space from it
    - $1/4^\text{th}$, $1/8^\text{th}$, ... in power of two
  - Ensures node can forward lookup query to at least $1/2$ of the remaining ID-space distance to key
  - Lookups in $O(\log N)$

Mapping of data items to nodes in Chord

Storing keys and forwarding lookups
- An entity with key $k$ falls under the **jurisdiction** of node with the smallest identifier $id$
  - $id \geq k$
  - Referred to as the successor of $k$ or $\text{successor}(k)$
- A node forwards query for key $k$ to node (in its FT) with highest $ID < k$
Chord lookup example for k=54

When a node wants to join
- Generate a random id
- Probability of collisions is low
- `lookup(id)`
  - Will return successor(id)
- Contact successor(id) and its predecessor
  - Insert self in the ring
  - Transfer data items
    - All keys must be fetched from the new node’s successor

An example of inserting a new node

Finger Table in Chord
- Chord uses an m-bit identifier space
  - 2^m possible peers
- Each node, p, in Chord maintains a Finger Table with m-entries
  - \( FT_p[i] = \text{succ}(p + 2^i) \)
A node is responsible for keys that fall in the range
key > predecessor
key <= self

To lookup a key k, node p will forward query to node q with index j in p's FT where:

- If highest ID less than k
  \[ q = \text{FT}_p[j] \leq k < \text{FT}_p[j+1] \]
- OR
  \[ q = \text{FT}_p[1] \text{ when } p < k < \text{FT}_p[1] \]

Stop forwarding the query when you are the target node

- A node is responsible for keys that fall in the range
  key > predecessor
  key <= self
Keeping the finger table up-to-date:

At node \( q \), \( FT_q[1] \) must be accurate

1. Contact \( \text{succ}(q+1) \) (This is \( FT_q[1] \))
   - Have it return its predecessor
2. If \( q = \text{pred}(\text{succ}(q+1)) \)
   - Everything is fine
3. Otherwise:
   - There is a new node \( p \) such that \( q < p < \text{succ}(q+1) \)
   - \( FT_q[1] = p \)
   - Check if \( p \) has recorded \( q \) as its predecessor
   - No? Go to step (1)

An example of inserting a new node \( N-4 \):
Node-4 comes in and contacts Node-1

Installing successor at Node-1

Updating the FT at \( N-1 \)
An example of inserting a new node N-7:

N-7 contacts N-1 for filling its FT

N-7 informs N-1 that it (N-7) is now N-1’s predecessor

Since 7 is closer it is installed as the predecessor of 1

When N-1 updates its FT later on ...

N-4 contacts N-1 to see if it is still its predecessor ... and installs N-7 as its successor

When the FT at N-4 is updated ...

UNSTRUCTURED P2P SYSTEMS
Structured P2P systems [Summary]

- Overall global policy governing
  - Topology of the network
  - Placements of objects
  - Routing functions to locate objects
- There is a specific distributed data structure that underpins
  - Associated Overlay
  - Algorithms that operate on it to route messages

Structured P2P systems [Summary]

- Because of the structure, algorithms are
  - Efficient
  - Offer time-bounds on object location
- BUT involve costly maintenance of underlying structures
  - In highly dynamic environments

Unstructured P2P systems [1/2]

- Target the maintenance argument
- No overall control on
  - Topology
  - Placements of objects within the network
- Overlay is created in an ad hoc manner
  - Each node joins network by following simple, local rules to establish connectivity

Unstructured P2P systems [2/2]

- A new joining node will establish contact with a set of neighbor nodes
  - These neighbors will be connected to further neighbors, etc.
- The network is fundamentally decentralized and self-organizing
  - Resilient to failures

Locating objects in unstructured P2P systems

- Requires a search of the resultant network topology
- No guarantees of being able to find the object
  - Performance will also be unpredictable
  - There is a risk of generating excessive message traffic to locate objects

Pros and Cons

<table>
<thead>
<tr>
<th></th>
<th>Structured P2P systems</th>
<th>Unstructured P2P systems</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Advantages</strong></td>
<td>Guaranteed to locate objects with bounds on this operation. Low message overhead</td>
<td>Self-organizing and naturally resilient to failures</td>
</tr>
<tr>
<td><strong>Disadvantages</strong></td>
<td>Maintain complex overlay structures that are difficult and costly in dynamic settings</td>
<td>Probabilistic Cannot offer absolute guarantees on locating objects</td>
</tr>
</tbody>
</table>
The contents of this slide-set are based on the following references