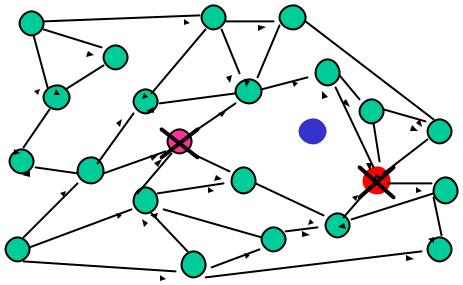
Changing the Tapestry—Inserting and Deleting Nodes

Kris Hildrum, UC Berkeley
hildrum@eecs.berkeley.edu

Joint work with John Kubiatowicz, Satish Rao, and Ben Zhao

Tapestry with Inserts and Deletes



Outline

- Insert
  - Finding surrogates
  - Constructing Neighbor tables
- Delete
- Unplanned Delete

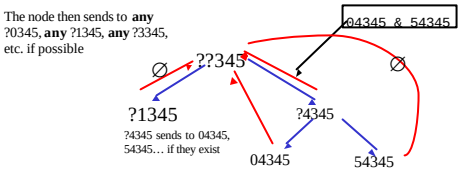
Requirement for Insert and Delete

- Use no central directory
  - No hot spot/single point of failure
  - Reduce danger/threat of DoS.
- Must be fast/touch few nodes
- Minimize system administrator duties
- Keep objects available

Acknowledged Multicast Algorithm

Locates & Contacts all nodes with a given suffix

- Create a tree based on IDs as we go
- Starting node knows when all nodes reached

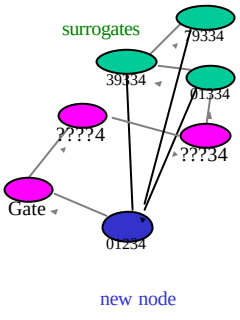


Three Parts To Insertion

1. Establish pointers from surrogates to new node.
2. Notify the need-to-know nodes
3. Create routing tables & notify other nodes

Finding the surrogates

- The new node sends a join message to a surrogate
- The primary surrogate multicasts to all other surrogates.
- Each surrogate establishes a pointer to the new node.
- When all pointers established, continue



Need-to-know nodes

- Need-to-know = a node with a hole in neighbor table filled by new node
  - If 01234 is new node, and no 234s existed, must notify ???34 nodes
  - Acknowledged multicast to all matching nodes
- During this time, object requests may go either to new node or former surrogate, but that's okay
- Once done, delete pointers from surrogates.

Constructing the Neighbor Table via a nearest neighbor search

- Suppose we have a good algorithm **A** for finding the three nearest neighbors for a given node.
- To fill in a slot, apply **A** to the subnetwork of nodes that could fill that slot.
  - For ????1, run **A** on network of nodes ending in 1
- Can do something more that requires less computation, but uses nearest neighbor.

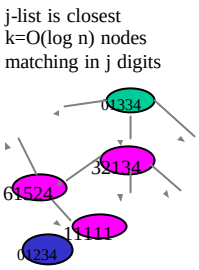
1/17/01

Finding Nearest Neighbor

- Let j be such that surrogate matches new node in last j digits of node ID
- G = surrogate

A. G sends j-list to new node; new node pings all nodes on j-list.

B. If one is closer, G = closest, goto A. If not, done with this level, and let j = j-1 and goto A.

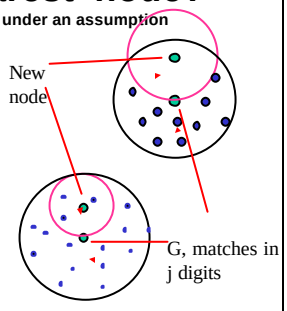


1/17/01

Is this the nearest node?

Yes, with high probability under an assumption

- Pink circle = ball around new node of radius d(G, new node)
- Progress = find any node in pink circle
- Consider the ball around the G containing all its j-list. Two cases:
  - Black ball contain pink ball; found closest node
  - High overlap between pink ball and G-ball so unlikely pink ball empty while G-ball has k nodes



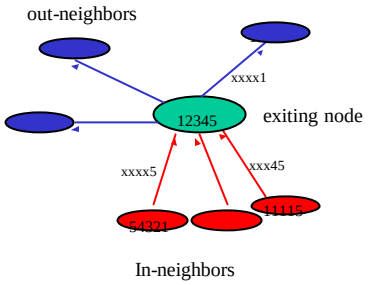
1/17/01

The Grid-like assumption

- The algorithm for finding the first entry works for any grid-like network
- Same as the assumption that Plaxton, Rajaraman, and Richa make.

1/17/01

Delete - Terminology



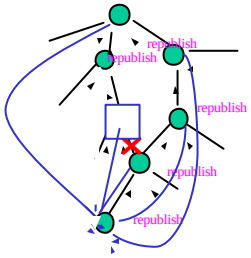
1/17/01

Planned Delete

- Notify its neighbors ($O(\log^2 n)$)
  - To out-neighbors: Exiting node says "I'm no longer pointing to you"
  - To in-neighbors: Exiting node says it is going and proposes at least one replacement.
  - Exiting node republishes all objects ptrs it stores
  - Use republish-on-delete to clean things up
- Objects rooted at exiting node get new roots
  - Either proactive pointer copying, or
  - wait for republishes and mean time, switch routing planes.

1/17/01

Republish-On-Delete



1/17/01

Unplanned Delete

- Planned delete relied exiting node's neighbor table.
  - List of out-neighbors
  - List of in-neighbors
  - Closest matching node for each level.
- Can we reconstruct this information?
  - Not easily
  - Fortunately, we probably don't need to.

1/17/01

Handle Unplanned Delete Lazily

- A notices B is dead, A fixes its own state
  - A removes B from routing tables
    - If removing B produces a hole, A must fill the hole, or be sure that the hole cannot be filled—use acknowledged multicast
  - A republishes all objs with next hop = B.
    - Use republish-on-delete as before
- Good: Each node makes a local decision, so no DoS problems.
- Problems
  - Delete may never "finish" and new nodes may get outdated information.
  - Partial delete undetected.

Conclusion – Insert and Delete works!

- No central point of failure
- Touches only polylog n nodes.
- Minimizes system administrator duties
- Objects always available