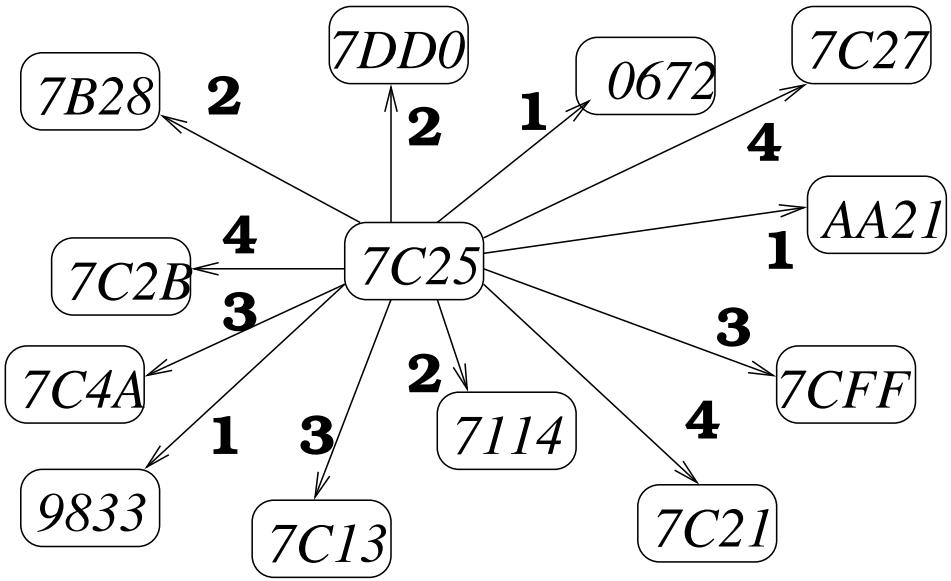
Tapestry

- Nodes and objects are assigned IDs from common space via a distributed hashing.
- Hashed node ids are termed VIDs or v_{id}. Hashed object identifiers are termed GUIDs or O_G.
- ID space typically has $m = 160$ bits, and is expressed in hexadecimal.
- If a node v exists such that $v_{id} = O_G$ exists, then that v become the root. If such a v does not exist, then another unique node sharing the largest common prefix with O_G is chosen to be the *surrogate root*.
- The object O_G is stored at the root, or the root has a direct pointer to the object.
- To access object O, reach the root (real or surrogate) using prefix routing
- Prefix routing to select the next hop is done by increasing the prefix match of the next hop's VID with the destination O_{G_R}. Thus, a message destined for $O_{G_R} = 62C35$ could be routed along nodes with VIDs 6****, then 62***, then 62C**, then 62C3*, and then to 62C35.

Tapestry - Routing Table

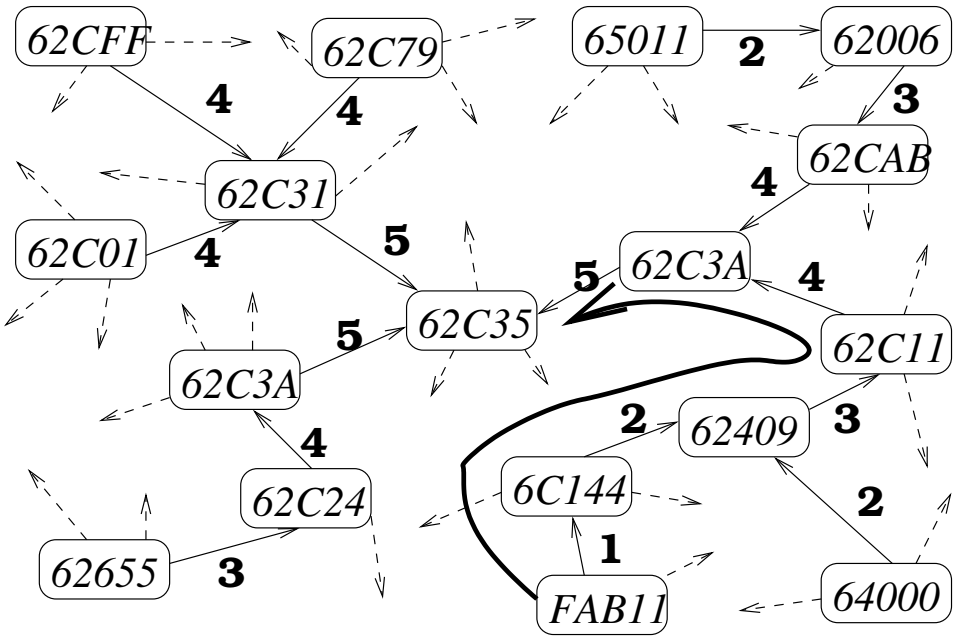
- Let $M = 2^m$. The routing table at node v_{id} contains $b \cdot log_b M$ entries, organized in $log_b M$ levels $i = 1 \ldots log_b M$. Each entry is of the form $\langle w_{id}, IP\ address \rangle$.
- Each entry denotes some "neighbour" node VIDs with a $(i-1)$-digit prefix match with v_{id} – thus, the entry's w_{id} matches v_{id} in the $(i-1)$-digit prefix. Further, in level i, for each digit j in the chosen base (e.g., $0, 1, \ldots E, F$ when $b = 16$), there is an entry for which the i^{th} digit position is j.
- For each forward pointer, there is a backward pointer.



Some example links at node with identifier "7C25". Three links each of levels 1 through 4 are labeled.

Tapestry: Routing

- The j^{th} entry in level i may not exist because no node meets the criterion. This is a *hole* in the routing table.
- *Surrogate routing* can be used to route around holes. If the j^{th} entry in level i should be chosen but is missing, route to the next non-empty entry in level i, using wraparound if needed. All the levels from 1 to $log_b 2^m$ need to be considered in routing, thus requiring $log_b 2^m$ hops.



An example of routing from FAB11 to 62C35. The numbers on the arrows show the level of the routing table used. The dashed arrows show some unused links.

Tapestry: Routing Algorithm

- Surrogate routing leads to a unique root.
- For each v_{id}, the routing algorithm identifies a unique spanning tree rooted at v_{id}.

(variables)
array of array of integer $Table[1 \ldots log_b\, 2^m, 1 \ldots b]$; // routing table

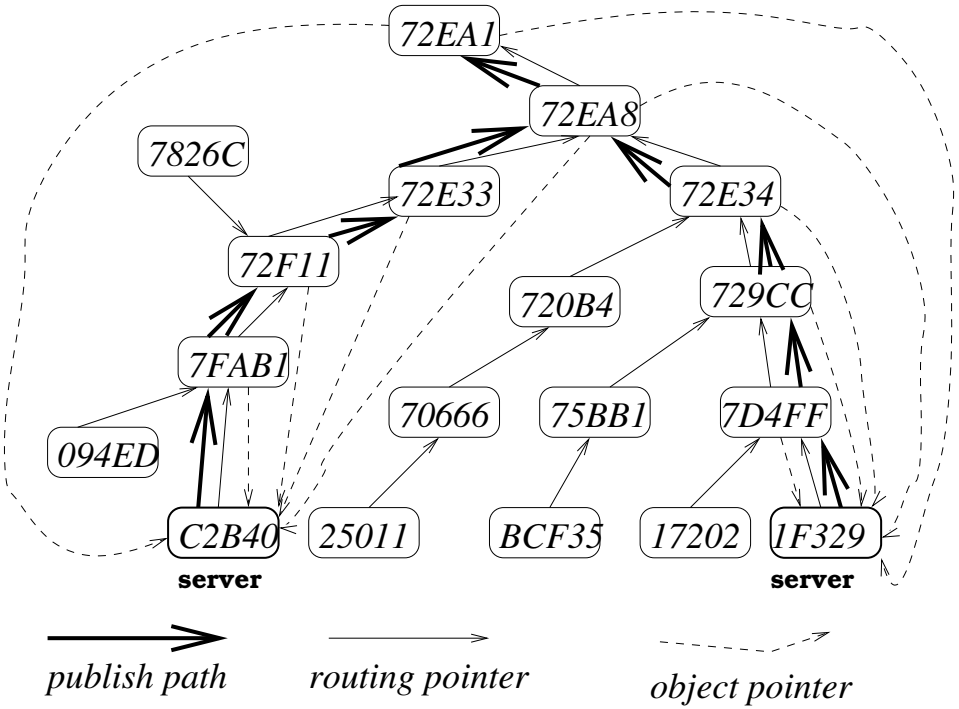
(1) $NEXT\_HOP(i, O_G = d_1 \circ d_2 \ldots \circ d_{log_b M})$ executed at node v_{id} to route to O_G:
// i is (1 + length of longest common prefix), also level of the table
(1a) **while** $Table[i, d_i] = \perp$ **do** // d_j is ith digit of destination
(1b) $d_i \longleftarrow (d_i + 1) \bmod b$;
(1c) **if** $Table[i, d_i] = v$ **then** // node v also acts as next hop (special case)
(1d) **return** $NEXT\_HOP(i + 1, O_G)$ // locally examine next digit of destination
(1e) **else return**($Table[i, d_i]$). // node $Table[i, d_i]$ is next hop

The logic for determining the next hop at a node with node identifier v, $1 \leq v \leq n$, based on the i^{th} digit of O_G.

Tapestry: Object Publication and Object Search

- The unique spanning tree used to route to v_{id} is used to publish and locate an object whose unique root identifier O_{G_R} is v_{id}.
- A server S that stores object O having GUID O_G and root O_{G_R} periodically publishes the object by routing a *publish* message from S towards O_{G_R}.
- At each hop and including the root node O_{G_R}, the *publish* message creates a pointer to the object
- This is the directory info and is maintained in *soft-state*.
- To search for an object O with GUID O_G, a client sends a query destined for the root O_{G_R}.
  - Along the $log_b 2^m$ hops, if a node finds a pointer to the object residing on server S, the node redirects the query directly to S.
  - Otherwise, it forwards the query towards the root O_{G_R} which is guaranteed to have the pointer for the location mapping.
- A query gets redirected directly to the object as soon as the query path overlaps the publish path towards the same root.

Tapestry: Object Publication and Search



An example showing publishing of object with identifier 72EA1 at two replicas 1F329 and C2B40. A query for the object from 094ED will find the object pointer at 7FAB1. A query from 7826C will find the object pointer at 72F11. A query from BCF35 will find the object pointer at 729CC.

Tapestry: Node Insertions

- For any node Y on the path between a publisher of object O and the root G_{O_R}, node Y should have a pointer to O.
- Nodes which have a hole in their routing table should be notified if the insertion of node X can fill that hole.
- If X becomes the new root of existing objects, references to those objects should now lead to X.
- The routing table for node X must be constructed.
- The nodes near X should include X in their routing tables to perform more efficient routing.

Refer to book for details of the insertion algorithm that maintains the above properties.

Tapestry: Node Deletions and Failures

Node deletion

- Node A informs the nodes to which it has (routing) backpointers. It also provides them with replacement entries for each level from its routing table. This is to prevent holes in their routing tables. (The notified neighbours can periodically run the nearest neighbour algorithm to fine-tune their tables.)
- The servers to which A has object pointers are also notified. The notified servers send object republish messages.
- During the above steps, node A routes messages to objects rooted at itself to their new roots. On completion of the above steps, node A informs the nodes reachable via its backpointers and forward pointers that it is leaving, and then leaves.

Node failures: Repair the object location pointers, routing tables and mesh, using the redundancy in the Tapestry routing network. Refer to the book for the algorithms

Tapestry: Complexity

- A search for an object expected to take $(log_b 2^m)$ hops. However, the routing tables are optimized to identify nearest neighbour hops (as per the space metric). Thus, the latency for each hop is expected to be small, compared to that for CAN and Chord protocols.
- The size of the routing table at each node is $c \cdot b \cdot log_b 2^m$, where c is the constant that limits the size of the neighbour set that is maintained for fault-tolerance.

The larger the Tapestry network, the more efficient is the performance. Hence, better if different applications share the same overlay.

Fairness in P2P systems

Selfish behavior, free-riding, leaching degrades P2P performance. Need incentives and punishments to control selfish behavior.

Prisoners' Dilemma

Two suspects, A and B, are arrested by the police. There is not enough evidence for a conviction. The police separate the two prisoners, and separately, offer each the same deal: if the prisoner testifies against (betrays) the other prisoner and the other prisoner remains silent, the betrayer gets freed and the silent accomplice get a 10 year sentence. If both testify against the other (betray), they each receive a 2 year sentence. If both remain silent, the police can only sentence both to a small 6-month term on a minor offense.

Rational selfish behavior: both betray the other - is not Pareto optimal (does not ensure max good for all). Both staying silent is Pareto-optimal but that is not rational. In the iterative version, memory of past moves can be used, in this case, Pareto-optimal solution is reachable.

Tit-for-tat in BitTorrent

Tit-for-tat strategy: first step, you cooperate; in subsequent steps, reciprocate the action done by the other in the previous step.

- The BitTorrent P2P system has adopted the tit-for-tat strategy in deciding whether to allow a download of a file in solving the leaching problem.
- cooperation is analogous to allowing others to upload local files,
- betrayal is analogous to not allowing others to upload.
- *chocking* refers to the refusal to allow uploads.

As the interactions in a P2P system are long-lived, as opposed to a one-time decision to cooperate or not, *optimistic unchocking* is periodically done to unchoke peers that have been chocked. This optimistic action roughly corresponds to the re-initiation of the game with the previously chocked peer after some time epoch has elapsed.