## TAPESTRY

***Tapestry is a peer-to-peer overlay network which provides a distributed hash table, routing, and multicasting infrastructure for distributed applications. The Tapestry peer-to-peer system offers efficient, scalable, self-repairing, location-aware routing to nearby resources.***

- Tapestry is a decentralized distributed system.
- It is an overlay network that implements simple key-based routing.
- It is a prototype of a decentralized, scalable, fault-tolerant, adaptive location and routing infrastructure
- Each node serves as both an object store and a router that applications can contact to obtain objects.
- In a Tapestry network, objects are published at nodes, and once an object has been successfully published, it is possible for any other node in the network to find the location at which that object is published.
- The difference between Chord and Tapestry is that in Tapestry the application chooses where to store data, rather than allowing the system to choose a node to store the object at.
- The application only publishes a reference to the object.
- The Tapestry P2P overlay network provides efficient scalable location independent routing to locate objects distributed across the Tapestry nodes.
- The hashed node identifiers are termed **VIDs** (Virtual ID) and the hashed object identifiers are termed as **GUIDs** (Globally Unique ID).

### Routing and Overlays

***Routing and overlay are the terms coined for looking objects and nodes in any distributed system.***

- It is a middleware that takes the form of a layer which processes the route requests from the clients to the host that holds the objects.

- The objects can be placed and relocated without the information from the clients.

**Functionalities of routing overlays**:

- A client requests an object with GUID to the routing overlay, which routes therequest to a node at which the object replica resides.
- A node that wishes to make the object available to peer-to-peer service computes the GUID for the object and announces it to the routing overlay that ensures that the object is reachable by all other clients.
- When client demands object removal, then the routing overlays must make them unavailable.
- Nodes may join or leave the service.

**Routing overlays in Tapestry**

- Tapestry implements Distributed Hash Table (DHT) and routes the messages to thenodes based on GUID associated with resources through prefix routing.
- **Publish (GUID**) primitive is issued by the nodes to make the network aware of its possession of resource.
- Replicated resources also use the same publish primitive with same GUID. This results in multiple routing entries for the same GUID.
- This offers an advantage that the replica of objects is close to the frequent users toavoid latency, network load, improve tolerance and host failures.

**Roots and Surrogate roots**

- Tapestry uses a common identifier space specified using m bit values and presently Tapestry recommends m = 160.
- Each identifier $O_G$ in this common overlay space is mapped to a set of unique nodes that exists in the network, termed as the identifier's root set denoted $O_{GR.}$
- If there exists a node v such that $v_{id} = O_{GR,}$ then v is the root of identifier $O_G$.

- If such a node does not exist, then a globally known deterministic rule is used to identify another unique node sharing the largest common prefix with $O_{G}$, that acts as the surrogate root.
- To access object O, the goal is to reach the root $O_{GR}$.
- Routing to $O_{GR}$ is done using distributed routing tables that are constructed using prefix routing information.

**Prefix Routing**

***Prefix routing at any node to select the next hop is done by increasing the prefix match of the next hop's VID with the destination $O_{GR}$.***

- Let $M = 2^m$. The routing table at node vid 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 $<w_{id}$, IP address$>$.
- The following is the property of entry (b) at level i:

  Each entry denotes some neighbor node VIDs with an (i – 1) digit prefix match with v id . Further, in level i, for each digit j in the chosen base, there is an entry for which the ith digit position is j. The jth entry (counting from 0) in level i has value j for digit position i. Let an i digit prefix of vid be denoted as prefix (vid, i). Then the jth entry (counting from 0) in level i begins with an i-digit prefix prefix (vid,i– 1). j.

**Routing Table**

- The nodes in the router table at $v_{id}$ are the neighbors in the overlay, and these are exactly the nodes with which $v_{id}$ communicates.
- For each forward pointer from node v to v', there is a backward pointer from v' to v.
- There is a choice of which entry to add in the router table. The jth entry in level i can be the VID of any node whose i-digit prefix is determined; the (m – i) digit suffix can vary.

- The flexibility is useful to select a node that is close, as defined by some metric space.
- This choice also allows a more fault-tolerant strategy for routing.
- Multiple VIDs can be stored in the routing table.
- The jth 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 jth 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.

```
(variables)
Integer Table[1…$\log_b 2^m$, 1….b];                //routing table
(1) NEXT_HOP(i, OG = $d_1$ o $d_2$ … o $d_{logbm}$) 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 the ith digit of destination
(1b)      $d_i \leftarrow (d_i+1)$ mod b;
(1c)  if Table[i, $d_i$] = v then            // node v also acts as next hop
                                                     // (special case)
(1d)      return (NEXT_HOP(i+1, OG) // locally examine next digit of
                                                     //destination
(1e)  else return (Table[i, $d_i$]).          // node Table[i, $d_i$] is next hop
```

**Fig : NEXT_HOP(i, $O_G$)**

**Object Publication and object searching**

- The unique spanning tree used to route to vid is used to publish and locate an object whose unique root identifier $O_{GR}$ is $v_{id}$.
- A server S that stores object O having GUID $O_G$ and root $O_{GR}$ periodically publishes the object by routing a publish message from S towards $O_{GR.}$

- At each hop and including the root node $O_{GR}$, the publish message creates a pointer to the object.
- Each node between O and $O_{GR}$ must maintain a pointer to O despite churn.
- If a node lies on the path from two or more servers storing replicas, that node will store a pointer to each replica, sorted in terms of a distance metric.
- This is the directory information for objects, and is maintained as a soft-state, i.e., it requires periodic updates from the server, to deal with changes and to provide fault-tolerance.
- To search for an object O with GUID $O_G$, a client sends a query destined for the root $O_{GR}$
- 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 $O_{GR}$ 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.
- Each hop towards the root reduces the choice of the selection of its next node by a factor of b; hence, the more likely by a factor of b that a query path and a publish path will meet.
- As the next hop is chosen based on the network distance metric whenever there is a choice, it is observed that the closer the client is to the server in terms of the distance metric, the more likely that their paths to the object root will meet sooner, and the faster the query will be redirected to the object.

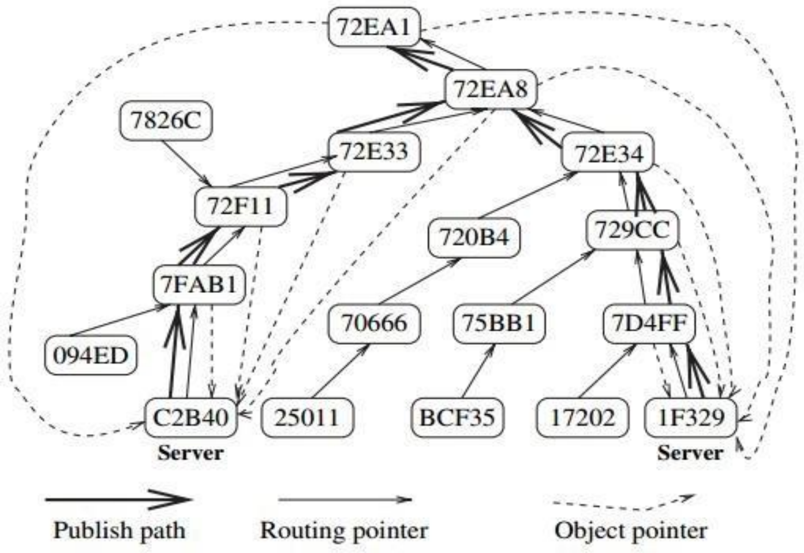

**Fig : Publishing of object with identifier 72EA1 at two replicas 1F329 and C2B40Node Insertion**

- When nodes join the network, the result should be the same as though the networkand the routing tables had been initialized with the nodes as part of the network.

- The procedure for the insertion of node X should maintain the following property of Tapestry: For any node Y on the path between a publisher of object O and the root $O_{GR}$, node Y should have a pointer to O.

**Properties for node insertion:**

- Nodes that 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 efficientrouting.

**Steps in insertion**

- Node X uses some gateway node into the Tapestry network to route a message toitself. This leads to its surrogate, i.e., the root node with identifier closest to that of itself (which is Xid). The surrogate Z identifies the length $\alpha$ of the longest common prefix that Zid shares with Xid.

- Node Z initiates a MULTICAST-CONVERGECAST on behalf of X by creating a logical spanning tree as follows. Acting as a root, Z contacts all the $(\alpha, j)$ nodes, for all $j \in \{0, 1, \ldots, b - 1\}$.

- These are the nodes with prefix $\alpha$followed by digit j. Each such (level 1) node Z1 contacts all the prefix $((Z1, |\alpha| + 1),j)$ nodes, for all $j \in \{0, 1,\ldots,b - 1\}$. This continues up to level $\log_b 2^m$ and completes the MULTICAST.
- The nodes at this level are the leaves of the tree, and initiate the CONVERGECAST, which also helps to detect the termination of this phase.
- The insertion protocols are fairly complex and deal with concurrent insertions.

**Node Deletion**

When a node A leaves the Tapestry overlay:

1. Node A informs the nodes to which it has back pointers. It also provides them with replacement entries for each level from its routing table. This is to prevent holes in their routing tables.
2. The servers to which A has object pointers are also notified. The notified servers send object republish messages.
3. During the above steps, node A routes messages to objects rooted at itself to theirnew roots. On completion of the above steps, node A informs the nodes reachable via its back pointers and forward pointers that it is leaving, and then leaves.

- Node failures are handled by using the redundancy that is built in to the routing tables and object location pointers.
- A node X detects a failure of another node A by using soft-state beacons or when a node sends a message but does not get a response.
- Node X updates its routing table entry for A with a suitable substitute node, running the nearest neighbor algorithm if necessary.
- If A's failure leaves a hole in the routing table of X, then X contacts the suggorate of A in an effort to identify a node to fill the hole.

- To repair the routing mesh, the object location pointers also have to be adjusted.
- Objects rooted at the failed node may be inaccessible until the object is republished.
- The protocols for doing so essentially have to:
  - i) maintain path availability
  - ii) optionally collect garbage/dangling pointers that would otherwise persist until the next soft-state refresh and timeout

**Complexity**

- A search for an object is expected to take $\log_b 2^m$ hops. The routing tables are optimized to identify nearest neighbor hops.
- 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 neighbor set that is maintained for fault-tolerance.