# *Tapestry*

G.T.K.Devaroy(07CS1012), K.Bala Kishan(07CS1024), Rahul(07CS3009)

September 9, 2010

## 1 Introduction

Tapestry is an overlay location and routing infrastructure that provides location-independent routing of messages directly to the closest copy of an object or service using only point-to-point links and without centralized resources. The routing and directory information within this infrastructure is purely soft state and easily repaired. Tapestry is self-administering, fault- tolerant, and resilient under load.

## 2 Plaxton Mesh:

A distributed data structure optimized to support a network overlay for locating named objects and routing of messages to those objects. Forwarding overlay messages is routing.

1. Each node can take the role of:
   - Servers: where objects are stored
   - Routers: which forward messages
   - Clients: Which is the origin of a request
2. Naming:
   - The Objects(Files) and nodes have independent names of their location and semantics.
   - The names are 40 Hex digits(160 bits) in size.
   - These names are computed using hashing algorithms.
3. Structuring:
   - Each node has its own routing table.
   - The routing table maintains levels and neighbours according to levels.
   - The size of routing table is Levels*Base. *example, 4*16 for 4 levels and base is HEX*

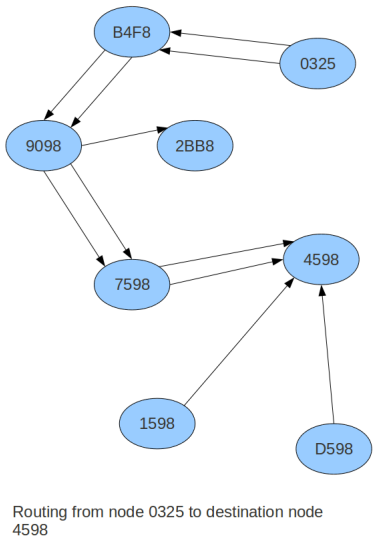


Figure 1: Routing Table of a node

## 2.1 Routing in Plaxton Mesh:

Plaxton uses local routing maps at each node, which we call neighbor maps, to incrementally route over-lay messages to the destination ID digit by digit (e.g.routing to destination node 4598 , ***8 ==> **98 ==> *598 ==> 4598 where *s represent don't cares).A node N has a neighbor map with multiple levels, where each level represents a matching suffix up to a digit position in the ID.

By definition, the $i'th$ node a message reaches shares a suffix of at least length i with the destination ID. To find the next router, we look at the $i+1'th$ level map, and look up the entry matching the value of the next digit in destination.

**Routing Procedure to a node:**

- If current node has shared prefix till i.
- Look at level i+1 in the table.
- Match the next digit in destination.
- Send message

Assuming consistent neighbor maps, this routing method guarantees that any existing unique node in the system will be found within at most $log_b N$ logical hops, in a system with an N size namespace using IDs of base b.

| | | | |
|---|---|---|---|
| 0642 | x042 | xx02 | xxx0 |
| 1642 | x142 | xx12 | xxx1 |
| 2642 | x242 | xx22 | xxx2 |
| 3642 | x342 | xx32 | xxx3 |
| 4642 | x442 | xx42 | xxx4 |
| 5642 | x542 | xx52 | xxx5 |
| 6642 | x642 | xx62 | xxx6 |
| 7642 | x742 | xx72 | xxx7 |

Figure 2: Routing Table of a node

## 2.2 Fault Tolerance:

Tapestry has the ability to detect, circumvent and recover from failures.Faults are an expected part of normal operation in the wide-area. Furthermore, faults are detected and circumvented by the previous hop router, minimizing the effect a fault has on the overall system.

### 2.2.1 Fault Tolerant Routing:

Failures can occur due to server outages(those due to high load and hardware/software failures), link failures (router hardware and software faults), neighbor table corruption at the server, and failure of intermediate nodes.

- Each entry table has two backup-ids(backup neighbours) apart from the primary neighbour.
- The Primary and back-up ID's are chosen based on RTT(Round Trip Time) to the neighbours.
- Whenever the Primary-ID fails, the backup ID's are initiated and a stream of control messages is passed to the failed primary neighbour to see if it is repaired.
- If the primary is repaired, then it is re-initiated.
- If the failed node is not repaired within a timeout interval, then the Secondary Neighbour is made primary and a new secondary node is brought in.

### 2.2.2 Fault Tolerant Location(*SALTING*):

Objects root node is a single point of failure in the Plaxton location mechanism.If the destination node fails, the object cannot be accessed as the information

about its location is lost. We correct this in Tapestry by assigning multiple roots to each object. To accomplish this, we concatenate a small, globally constant sequence of salt values (e.g. 1, 2, 3) to each object ID, then hash the result to identify the appropriate roots. These roots are used (via surrogate routing) during the publishing process to insert location information into the Tapestry. When locating an object, Tapestry performs the same hashing process with the target object ID, generating a set of roots to search.

## 2.3 Surrogates:

Surrogate routing tentatively chooses an objects root node to have the same name as its ID,I . Given the sparse nature of the node name space, it is unlikely that destination node for an object will actually exist. In such cases, SURROGATE nodes are used to store those objects.

In the original Plaxton scheme, an objects root or surrogate node is chosen as the node which matches the objects ID in the greatest number of trailing bit positions. Since there may be many nodes which match this criteria, the Plaxton scheme chooses a unique root by invoking a total ordering on all nodes in the network. The candidate node with the greatest position in this ordering is choosen as a root. Given this scheme, Plaxton location proceeds by resolving an objects ID one digit at a time until it encounters an empty neighbor entry. This node is used as a surrogate node for the object.

## 2.4 Handling New Node Entries:

Whenever a new node enters the network, the following things need to be taken care of:

- *Routing Table:* A new nodes needs its own routing table. So a new routing table has to be constructed.
- *Surrogates and Surrogate Mapping:* Nodes which were acting as surrogates to the new node till this point change. The following changes occur:
  - Step 1: Build up Ns routing maps.
    1. Send messages to each hop along path from gateway to current node N that best approximates N(N' is the surrogate for N).
    2. The $i'th$ hop along the path sends its $i'th$ level route table to N. All these table constitute the complete table for node N.
    3. N optimizes those tables where necessary.
  - Step 2: Move appropriate data from N' to N
  - Step 3: Use back pointers from N' to find nodes which have null entries for Ns ID, tell them to add new entry to N.
  - Step 4: Notify local neighbors to modify paths to route through N where appropriate

## 2.5 Benefits and Limitation:

### 2.5.1 Benefits:

- *Simple Fault Handling:* Since routing only requires nodes match a certain suffix, there is potential to route around any single link or server failure by choosing another node with a similar suffix.
- *Scalable:* It is inherently decentralized, and all routing is done using locally available data. Without a point of centralization, the only possible bottleneck exists at the root node.
- *Exploiting Locality:* With a reasonably distributed namespace, resolving each additional digit of a suf- fix reduces the number of satisfying candidates by a factor of the ID base b (the number of nodes that satisfy a suffix with one more digit specified decreases geometrically). The path taken to the root node by the publisher or server S storing O and the path taken by the client C will likely converge quickly, because the number of nodes to route to drops geometrically with each additional hop. Therefore, queries for local objects are likely to quickly run into a router with a pointer to the objects location.
- *Proportional Route:* Distance Plaxton has proven that the total network distance traveled by a mes- sage during both location and routing phases is proportional to the underlying network distance, assuring us that routing on the Plaxton overlay incurs a reasonable overhead.

### 2.5.2 Limitations:

The limitaions of Tapestry are as follows:

- *Root Node Vulnerability:* As a location mechanism, the root node for an object is a single point of failure because it is the node that every client relies on to provide an objects location information. While intermediate nodes in the location process are interchangeable, a corrupted or unreachable root node would make objects invisible to distant clients, who do not meet any intermediate hops on their way to the root.
- *Global Knowledge:* In order to achieve a unique mapping between document identifiers and root nodes, the Plaxton scheme requires global knowledge at the time that the Plaxton mesh is constructed. This global knowledge greatly complicates the process of adding and removing nodes from the network.
- *Lack of Ability to Adapt:* While the location mechanism exploits good locality, the Plaxton scheme lacks the ability to adapt to dynamic query patterns, such as distant hotspots. Correlated access pat- terns to objects are not exploited, potential trouble spots are not corrected before they cause overload or cause congestion problems over the wide-area. Similarly,

the static nature of the Plaxton mesh means that insertions could only be handled by using global knowledge to recompute the function for mapping objects to root nodes.