Tapestry Algorithm

Implementation of Tapestry Algorithm

1. Node ID generation

The Node IDs in the Tapestry implementation are generated using SHA-1 hash function in Elixir. They are globally unique identifiers with fixed 4-digit length sequences of base-16. For example, a node could have an ID "F10A". In the implementation, the maximum number of hops required to reach a node id for message passing is equal to 4 (log N with base β, maximum unique N values possible for 4-digit Hexadecimal = 65536, β = 16, log 65536 with base 16 = 4).

2. Routing and Node State

Every node has routing tables maintained up to date in its state, for it to route to a destination node id and traverse through the network.

2.1 Routing Table

In Tapestry network, since node IDs are of 4 digits, 4 levels are implemented in every node's routing table, i.e. *Level-0, Level-1, Level-2* and *Level-3*. The number of columns in the routing table is equal to the β value, which is 16 in this case.

Rows in a routing table represents the levels/ hops and columns represent digits. (0 to F, β=16). A routing table entry can be defined based on its level and digit. Any node on level *"i"* has shared prefix *"i"* with the local node. The digit in columns represents the first digit after the shared prefix, i.e. *node[i]* = digit_value and this value represents the slot/column in with the node must be places in local node's routing table. (*i* values start from 0, if prefix is 0 means there is no shared prefix with the local node and row for this node is *Level-0* and column is the *node[i]* value). Only one node is placed per slot in the tapestry implementation.

Level-0 contains the set of nearest nodes starting with 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F and gets filled in the columns based on their first digit values as shown below in the routing table. Nodes at this level do not share prefix values with the local node.

Level-1 contains the set of nearest nodes sharing *prefix* 1 with the local node ID and gets filled in the columns based on their second digit values in the local node's routing table.

Level-2 contains the nearest nodes sharing *prefix* 2 with the local node ID and gets filled in the columns based on their third digit values in the local node's routing table.

Level-3 contains the nearest nodes sharing *prefix* 3 with the local node ID and gets filled in the columns based on their fourth digit values in the local node's routing table.

Nodes sharing same prefix value exhibit similar *"0"* entries in their respective levels, this property maintains consistency among the nodes in a Tapestry network. An example is shown below.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
L0	**0**FDD	**1**FC8	**2**DFC	**3**011	**4**050	**5**007	**6**005	**7**013	**8**01B	**9**028	0	**B**019	**C**032	**D**00D	**E**01B	**F**001
L1	2**0**F9	0	2**2**FD	2**3**ED	2**4**5D	2**5**ED	0	2**7**E7	2**8**B5	2**9**A2	2**A**A0	2**B**EE	2**C**D6	2**D**FC	2**E**51	2**F**10
L2	0	0	0	0	0	0	0	0	0	0	2D**A**D	2D**B**B	0	0	2D**E**B	2D**F**C
L3	0	0	0	0	0	0	0	0	0	0	0	2DE**B**	0	0	0	0

Table 1: Routing table of **2DEB**

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
L0	**0**FDD	**1**FC8	**2**DEB	**3**011	**4**050	**5**007	**6**005	**7**013	**8**01B	**9**028	0	**B**019	**C**032	**D**00D	**E**01B	**F**001
L1	2**0**F9	0	2**2**FD	2**3**ED	2**4**5D	2**5**ED	0	2**7**FC	2**8**B5	2**9**A2	2**A**A0	2**B**EE	2**C**D6	2**D**EB	2**E**51	2**F**10
L2	0	0	0	0	0	0	0	0	0	0	2D**A**D	2D**B**B	0	0	2D**E**B	2D**F**C
L3	0	0	0	0	0	0	0	0	0	0	0	0	2DF**C**	0	0	0

Table 2: Routing table of **2DFC**

Nodes **2DEB** and **2DFC** have prefix 2, and display consistency by populating *level 0, 1, 2* slots with *"0"* wherever there is no matching entry found. Suppose, there is a suitable node ID in a *level 2* slot for node **2DEB**, in order to avoid inconsistencies, a node entry should be present in **2DFC**'s *level 2* slot. Therefore, the *"0"* values at empty slots from *level 0* to *level prefix* (in this case *prefix* = 2) matches for **2DEB** and **2DFC.**

2.2 Backpointers Table

While creating the routing table for a local node, nearest neighbors are added into the routing table based on the matching prefix and digit following matching prefix values. Backpointer tables contain exact opposite information, where every local node saves its back pointers or node whose routing tables have local node's reference in their routing tables. For instance, in *table 1* **2DEB** has **23ED** in its routing table (*level 1* digit 3), so **23ED** will store **2DEB** as one of its backpointers. Therefore, when **23ED** is accessed in backpointer table, it returns all the nodes that had references to **23ED** in their routing tables. This implementation is used while adding a node dynamically to a network.

3. Dynamic Node Insertion

3.1 Node join

When a node is inserted into the network, we first find the root node (the node in the network with maximum prefix digits matching) for the inserted node. Then, access the root node's routing table and compute the prefix match of the inserted node and its root node. Based on the prefix value, the corresponding level is accessed from the root node's routing table and added to the new nodes routing table as part of *node join*. Backpointers of the neighbor nodes present in the *level (prefix)* are obtained and unique node values are used to populate the *level* (*prefix-1*) of the inserted node, this is done recursively until *level 0* is reached. The *level* (*prefix+1*) consists of *"0"* values because the maximum prefix node for

inserted node is the root node. The inserted node's routing table is populated with values from the root node's routing table and their backpointers. Additionally, a multicast message is sent across the network informing new node's insertion. In this process, nodes close to inserted node may consider adding the new node to their routing tables as an optimization. An example is shown below.

```
Node inserted: F4CB, Root node: F469
-------Node join: Routing table -> level: 0 to (prefix ) -------
[
  ["0", "1CE8", "2E97", "3113", "0", "5B68", "68C9", "7F0F", "0", "90A1",
   "A007", "0", "CD0E", "0", "EEAB", "F469"],
  ["0", "0", "0", "0", "F469", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0",
   "0"],
  ["0", "0", "0", "0", "0", "0", "F469", "0", "0", "0", "0", "0", "0", "0", "0",
   "0"]
]
-------Node join: Routing table for F4CB -------
[
  ["0", "1CE8", "2E97", "3113", "0", "5B68", "68C9", "7F0F", "0", "90A1",
   "A007", "0", "CD0E", "0", "EEAB", "F469"],
  ["0", "0", "0", "0", "F469", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0",
   "0"],
  ["0", "0", "0", "0", "0", "0", "F469", "0", "0", "0", "0", "0", "0", "0", "0",
   "0"],
  ["0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0",
   "0"]
]
---------------<>---------------
```

Example 1: Populating routing table for **F4CB** using **F469** as root node as a part of node join

```
"F469"
[
  ["0", "1CE8", "2E97", "3113", "0", "5B68", "68C9", "7F0F", "0", "90A1",
   "A007", "0", "CD0E", "0", "EEAB", "F469"],
  ["0", "0", "0", "0", "F469", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0",
   "0"],
  ["0", "0", "0", "0", "0", "0", "F469", "0", "0", "0", "0", "0", "0", "0", "0",
   "0"],
  ["0", "0", "0", "0", "0", "0", "0", "0", "0", "F469", "0", "0", "0", "0", "0",
   "0"]
]
```

Example 1: Routing table for **F469** before dynamic node join of **F4CB**

```
-------Node join: F4CB, updating routing table of node F469-------
[
  ["0", "1CE8", "2E97", "3113", "0", "5B68", "68C9", "7F0F", "0", "90A1",
   "A007", "0", "CD0E", "0", "EEAB", "F4CB"],
  ["0", "0", "0", "0", "F469", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0",
   "0"],
  ["0", "0", "0", "0", "0", "0", "F469", "0", "0", "0", "0", "0", "F4CB", "0",
   "0", "0"],
  ["0", "0", "0", "0", "0", "0", "0", "0", "0", "F469", "0", "0", "0", "0", "0",
   "0"]
]
```

Example 1: Updating routing table of **F469** after dynamic node join of **F4CB**

3.2 Surrogate Routing

This concept is used in node insertion when a node is connected to an existing Tapestry network. In order to find the root node of a new node, a message can be passed from any random node addressing to the new node's node id. While passing message to a destination node (in this case new nodes node id), when the destination node id is not present in the network, the message gets delivered to the nearest node, which shares maximum prefix and minimum distance from of the destination node. The node is known as surrogate node, and the routing is known as surrogate routing.

3.3 Root Node

Every node is the root of itself. If a node is dynamically added to an existing network, the nearest node sharing maximum prefix is the root node for the newly added node. The root node is used in case of node insertion and node deletion. In case of node insertion, in order to find the root node, it first searches for all the nodes starting with the first digit of the new node ID in the network. Then, it checks for the list of nodes sharing first two digits with the new node. If not found, it looks for the nodes sharing same first digit as new node and having *digit+1* value in their second digit. When a list of nodes sharing above mentioned prefix are encountered, it checks for the nearest one by comparing them to the new node ID. This goes on till it finds the root node.

4. Message Passing using prefix routing

If we want to pass the message from a source node id to the destination node id, the source node first checks the *level 0* of its routing table and looks for the entry against the column with *first digit* that matches the first digit of the destination node id. The message is forwarded to the intermediate node obtained from the source node's routing table as mentioned above. Upon accessing the intermediate node's routing table, number of hops is equated to be one and so row = *level 1* and column = *second digit* of destination node id are accessed to obtain the second intermediate node traversed along the path to the destination node id. While doing this, it maintains the condition that the first digit and second digit values are matching with the destination node's ID (*prefix* = 2, *hops* = 2 by the time it reaches next intermediate node). While traversing across routing tables running into a case where the slot value is *"0"* is possible and during such a scenario, the node present in the same row's next column (*column + 1*) is considered as a suitable intermediate node. This goes on until either the destination node is reached within 4 hops or stops at the 4^{th} hop and reaches a surrogate node. Usually, when there are no node failures or network failures, every node in the same network is reachable from any another node within a 4 hops count.

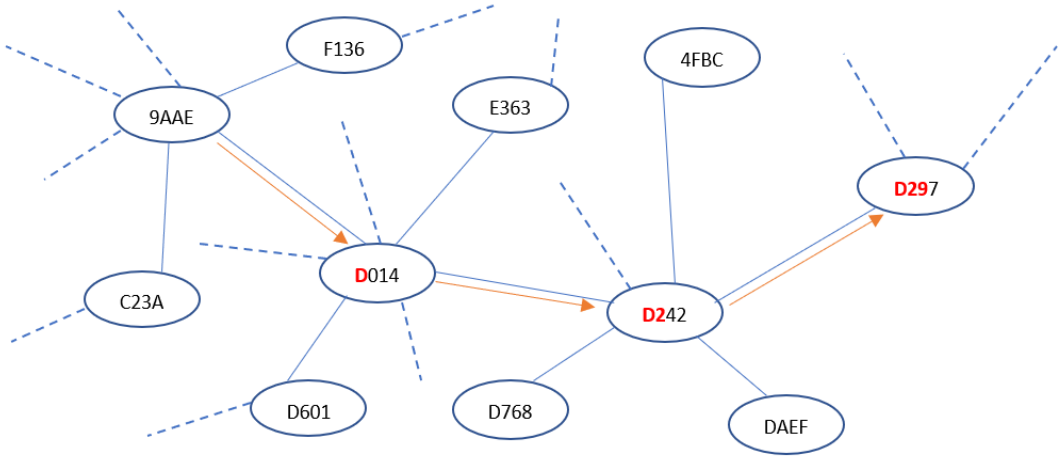


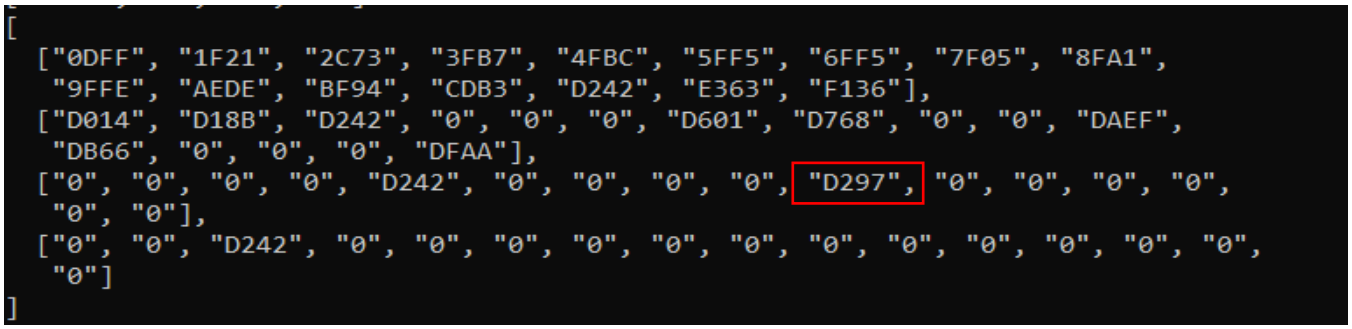
Figure 4: Message passing request from node **9AAE** (source node) to **D297** (destination node)

```
[
  ["0DFF", "1F21", "2C73", "3FB7", "4FBC", "5FF5", "6FF5", "7F05", "8FA1",
   "9AA6", "A12A", "B042", "C23A", "D014", "E363", "F136"],
  ["0", "919A", "0", "9376", "9462", "0", "0", "0", "98CE", "9972", "9AA6",
   "9B28", "0", "0", "0", "9FFE"],
  ["0", "0", "0", "0", "0", "9A5A", "0", "0", "0", "0", "9AA6", "0", "0", "0",
   "0", "0"],
  ["0", "0", "0", "0", "0", "0", "9AA6", "0", "0", "0", "0", "0", "0", "0", "0",
   "0"]
]
```

Example 2: In **9AA6**'s routing table, selected **D014** as the next hop for passing message from **9AA6** to **D297** (*hop count = 1*)

```
[
  ["0DFF", "1F21", "2C73", "3FB7", "4FBC", "5FF5", "6FF5", "7F05", "8FA1",
   "9FFE", "AEDE", "BF94", "CDB3", "D014", "E363", "F136"],
  ["D014", "D18B", "D242", "0", "0", "0", "D601", "D768", "0", "0", "DAEF",
   "DB66", "0", "0", "0", "DFAA"],
  ["0", "D014", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0",
   "0"],
  ["0", "0", "0", "0", "D014", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0",
   "0"]
]
```

Example 2: In **D014**'s routing table, selected **D242** as the next hop for passing message from **9AA6** to **D297** (*hop count = 2*)



Example 2: In **D242**'s routing table, selected **D297** as the next hop for passing message from **9AA6** to **D297** (*hop count = 3*)

Reached the destination node **D297** from source node **9AA6** in 3 hops.

Observations for message passing in Tapestry Algorithm

1. It has been observed that the value of "numRequests" influences the maximum hop count. When number of requests from peer to peer increases, probability of running into scenarios where the destination node ID (nodes that are far without matching prefix) takes 4 hops also increases.
2. When nodes in a network are increasing, the hop count also increases.
3. The above observation (2) is made in 2 scenarios, first one when the destination ID and source ID do not share common neighboring nodes in their respective routing tables.
4. Second scenario is when the slots in a node's routing table are not able to accommodate all the nodes sharing similar prefix values which will in turn result in an increase of hop count.
5. There is no surrogate routing observed in this implementation as all the unique node IDs are reachable from each other within 4 hops in the tapestry network.
6. The above performance (5) can be obtained by consistently updating and optimizing the routing tables during dynamic node insertions and opting nearest neighbor criteria to choose a node from a list of common prefix nodes while populating a slot in routing table.

Number of Nodes	NumRequests = 1	NumRequests = 3	NumRequests = 5
5	1	1	1
10	1	2	2
15	2	2	2
20	2	2	3
30	2	3	3
50	3	3	3
100	3	3	3
200	4	4	4
300	4	4	4
500	4	4	4
1000	4	4	4
2000	4	4	4

Table 3: Maximum number of hops for range of node values and peer to peer requests in tapestry network