Notebooks

y Routing

10:14 PM

+ Notebook

General Instruction

- Submit your work in the Dropbox folder via BeachBoard (Not email or in class).

1. (5 points) Fill out the following `google sheet`.
   - **Due by the Lab session (4/16 for S3, 4/17 for S5)**
   - Clink here to visit the `google sheet`
   - Find your `CSULB ID`.
   - Note your `Pastry ID`.
   - Write down your `AWS IPv4` into the `google sheet`.
2. (25 points) Implement a `Pastry` routing server in `Java` on your `AWS server`.
   - Read the section 10.5.1 carefully.
   - Use the Pastry IDs and IP addresses of the `google sheet`.
   - Build your own **leaf set** table as shown in Table 1.

Table 1: Pastry leaf set ($l = 2$) for `Pastry ID: 1230`. Each cell contains `ID:IP`.

SMALLER		LARGER	
1220:x.x.x.x	1223:x.x.x.x	1232:x.x.x.x	1241:x.x.x.x

   - Build your own **routing** table as shown in Table 2.

Table 2: Pastry routing table for `Pastry ID: 1230`. Each cell contains `ID:IP`.

0-132:x.x.x.x	**1:NA**	2-012:x.x.x.x	3-320:x.x.x.x
10-31:x.x.x.x	11-23:x.x.x.x	**12:NA**	13-10:x.x.x.x
120-3:x.x.x.x	121-1:x.x.x.x	122-0:x.x.x.x	**123:NA**
1230:y.y.y.y	1231:NULL	1232:x.x.x.x	1233:NULL

     1. y.y.y.y is the IP of Pastry ID: 1230.
     2. For the first, second, and third row, choose a node randomly. For instance, the **prefix 11**, select one of nodes whose ID has the **common prefix 11**. It can be one of 1100, 1101, 1102, ..., and 1133.
   - Open inbound `UDP port` 32710 on the `AWS server`.
   - Refer Figure 4.3 and Figure 4.4 of the text book and the assignment 5.

- The program specification.
  1. It should use a `Map` data structure to store Table 1 and Table 2.
  2. It should use `UDP datagram` (port 32710) **NOT** `TCP stream`.
  3. It should **reply** the ID:IP address of a **request** Pastry ID.
  4. Request - reply examples
     1230 - 1230:y.y.y.y
     1231 - 1231:NULL
     0123 - 0132:x.x.x.x
     1030 - 1031:x.x.x.x
     1123 - 1123:x.x.x.x
     1210 - 1211:x.x.x.x
     1211 - 1211:x.x.x.x
     1212 - 1211:x.x.x.x
     1213 - 1211:x.x.x.x
     1241 - 1241:x.x.x.x (from Table 1)
- Test the server program by implementing a simple `UDP` client on your local machine.
- Submit the source codes of the server and client.
- Leave your `AWS` IP on the comment section.
- Make sure that the server program is running on your server after submitting your work by using the command `nohup java SERVER_PROGRAM &`.

1. Fill out the google sheet
   a. Find your ID
      i.

68	14567876	3333	
69	14587792	3013	13.56.191.118
70	14608072	0331	

   b. Note your Pastery ID:
      i. 3013
   c. Write down your AWS IPv4
      i. Check
2. Implement a Pastery routing server in Java on your AWS server.
   a. Read Section 10.5.1 carefullly
      i. K
   b. Use the Pastery IDs and IP address of the google sheet
   c. Build your own leaf set table as shown in Table 1
      i. Oh

the book

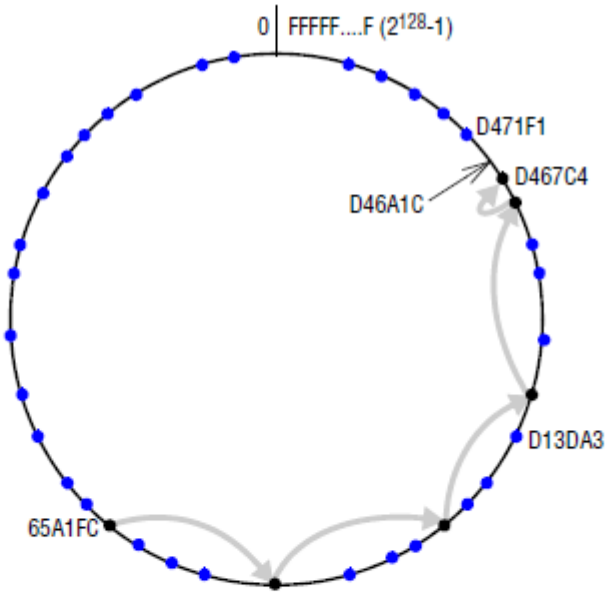
astry

astry [Rowstron and Druschel 2001, Castro *et al.* 2002a, freepastry.org] is a routing verlay with the characteristics that we outlined in Section 10.4. All the nodes and bjects that can be accessed through Pastry are assigned 128-bit GUIDs. For nodes, ese are computed by applying a secure hash function (such as SHA-1; see Section 1.4.3) to the public key with which each node is provided. For objects such as files, the UID is computed by applying a secure hash function to the object's name or to some art of the object's stored state. The resulting GUIDs have the usual properties of secure ash values – that is, they are randomly distributed in the range 0 to 2^{128}–1. They ovide no clues as to the value from which they were computed, and clashes between UIDs for different nodes or objects are extremely unlikely. (If a clash occurs, Pastry tects it and takes remedial action.)

In a network with N participating nodes, the Pastry routing algorithm will rrectly route a message addressed to any GUID in $O(\log N)$ steps. If the GUID entifies a node that is currently active, the message is delivered to that node; otherwise, e message is delivered to the active node whose GUID is numerically closest to it. ctive nodes take responsibility for processing requests addressed to all objects in their umerical neighbourhood.

Routing steps involve the use of an underlying transport protocol (normally UDP) transfer the message to a Pastry node that is 'closer' to its destination. But note that e closeness referred to here is in an entirely artificial space – the space of GUIDs. The

5 Circular routing alone is correct but inefficient *Based on Rowstron and Druschel [2001]*



The dots depict live nodes. The space is considered as circular: node 0 is adjacent to node (2^{128}-1). The diagram illustrates the routing of a message from node 65A1FC to D46A1C using leaf set information alone, assuming leaf sets of size 8 ($l = 4$). This is a degenerate type of routing that would scale very poorly; it is not used in practice.

real transport of a message across the Internet between two Pastry nodes may require a substantial number of IP hops. To minimize the risk of unnecessarily extended transport paths, Pastry uses a locality metric based on network distance in the underlying network (such as a hop counts or round-trip latency measurements) to select appropriate neighbours when setting up the routing tables used at each node.

Thousands of hosts located at widely dispersed sites can participate in a Pastry overlay. It is fully self-organizing: when new nodes join the overlay they obtain the data needed to construct a routing table and other required state from existing members in *O(log N)* messages, where *N* is the number of hosts participating in the overlay. In the event of a node failure or departure, the remaining nodes can detect its absence and cooperatively reconfigure to reflect the required changes in the routing structure in a similar number of messages.

Routing algorithm • The full routing algorithm involves the use of a routing table at each node to route messages efficiently, but for the purposes of explanation, we describe the routing algorithm in two stages. The first stage describes a simplified form of the algorithm that routes messages correctly but inefficiently without a routing table, and

Figure 10.7 First four rows of a Pastry routing table

p =	*GUID prefixes and corresponding node handles n*															
0	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*	*E*	*F*
	n	*n*	*n*	*n*	*n*	*n*		*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*
1	*60*	*61*	*62*	*63*	*64*	*65*	*66*	*67*	*68*	*69*	*6A*	*6B*	*6C*	*6F*	*6E*	*6F*
	n	*n*	*n*	*n*	*n*		*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*
2	*650*	*651*	*652*	*653*	*654*	*655*	*656*	*657*	*658*	*659*	*65A*	*65B*	*65C*	*65D*	*65E*	*65F*
	n	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*		*n*	*n*	*n*	*n*	*n*
3	*65A0*	*65A1*	*65A2*	*65A3*	*65A4*	*65A5*	*65A6*	*65A7*	*65A8*	*65A9*	*65AA*	*65AB*	*65AC*	*65AD*	*65AE*	*65AF*
	n		*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*

The routing table is located at a node whose GUID begins 65A1. Digits are in hexadecimal. The *n*s represent [GUID, IP address] pairs that act as node handles specifying the next hop to be taken by messages addressed to GUIDs that match each given prefix. Grey-shaded entries in the table body indicate that the prefix matches the current GUID up to the given value of *p*: the next row down or the leaf set should be examined to find a route. Although there are a maximum of 128 rows in the table, only $\log_{16} N$ rows will be populated on average in a network with *N* active nodes.

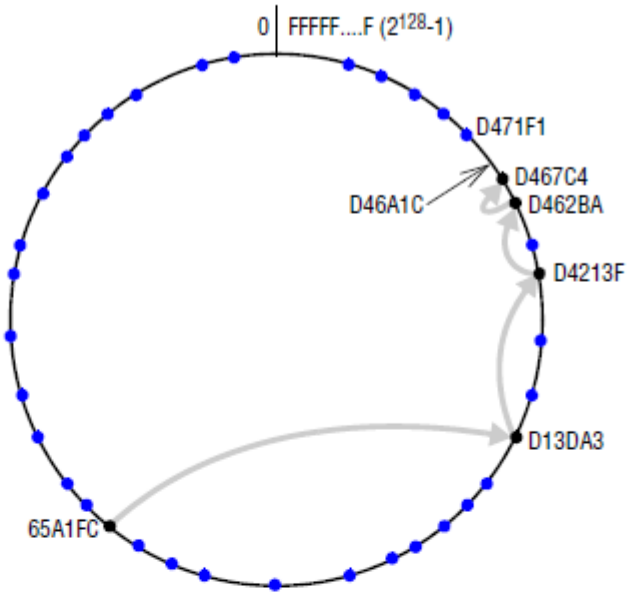
the second stage describes the full routing algorithm, which routes a request to any node in *O(log N)* messages:

Stage 1: Each active node stores a *leaf set* – a vector *L* (of size 2*l*) containing the GUIDs and IP addresses of the nodes whose GUIDs are numerically closest on either side of its own (*l* above and *l* below). Leaf sets are maintained by Pastry as nodes join and leave. Even after a node failure, they will be corrected within a short time. (Fault recovery is discussed below.) It is therefore an invariant of the Pastry system that the leaf sets reflect a recent state of the system and that they converge on the current state in the face of failures up to some maximum rate of failure.

The GUID space is treated as circular: GUID 0's lower neighbour is $2^{128}-1$.

Figure 10.6 gives a view of active nodes distributed in this circular address space. Since every leaf set includes the GUIDs and IP addresses of the current node's immediate neighbours, a Pastry system with correct leaf sets of size at least 2 can route messages to any GUID trivially as follows: any node A that receives a message M with destination address D routes the message by comparing D with its own GUID A and with each of the GUIDs in its leaf set and forwarding M to the node amongst them that is numerically closest to D.

Figure 10.8 Pastry routing example *Based on Rowstron and Druschel [2001]*



Routing a message from node 65A1FC to D46A1C. With the aid of a well-populated routing table the message can be delivered in ~ $\log_{16}(N)$ hops.

Figure 10.6 illustrates this for a Pastry system with $l = 4$. (In typical real installations of Pastry, $l = 8$.) Based on the definition of leaf sets we can conclude that at each step M is forwarded to a node that is closer to D than the current node and that this process will eventually deliver M to the active node closest to D. But such a routing scheme is clearly very inefficient, requiring ~ $N/2l$ hops to deliver a message in a network with N nodes.

Stage II: The second part of our explanation describes the full Pastry algorithm and shows how efficient routing is achieved with the aid of routing tables.

Each Pastry node maintains a tree-structured routing table giving GUIDs and IP addresses for a set of nodes spread throughout the entire range of 2^{128} possible GUID values, with increased density of coverage for GUIDs numerically close to its own.

Figure 10.7 shows the structure of the routing table for a specific node, and Figure 10.8 illustrates the actions of the routing algorithm. The routing table is structured as follows: GUIDs are viewed as hexadecimal values and the table classifies GUIDs based on their hexadecimal prefixes. The table has as many rows as there are hexadecimal digits in a GUID, so for the prototype Pastry system that we are describing, there are 128/4 = 32 rows. Any row n contains 15 entries – one for each possible value of the n^{th} hexadecimal digit, excluding the value in the local

Figure 10.9 Pastry's routing algorithm

To handle a message M addressed to a node D (where $R[p,i]$ is the element at column i, row p of the routing table):

```
1. If (L_-l < D < L_l) {  // the destination is within the leaf set or is the current node.
2.     Forward M to the element L_i of the leaf set with GUID closest to D or the
       current node A.
3. } else { // use the routing table to despatch M to a node with a closer GUID
4.     Find p, the length of the longest common prefix of D and A,. and i, the (p+1)th
       hexadecimal digit of D.
5.     If (R[p,i] ≠ null) forward M to R[p,i] // route M to a node with a longer common pre-
       fix.
6.     else { // there is no entry in the routing table.
7.         Forward M to any node in L or R with a common prefix of length p but a
           GUID that is numerically closer.
       }
   }
```

node's GUID. Each entry in the table points to one of the potentially many nodes whose GUIDs have the relevant prefix.

The routing process at any node A uses the information in its routing table R and leaf set L to handle each request from an application and each incoming message from another node according to the algorithm shown in Figure 10.9.

We can be sure that the algorithm will succeed in delivering M to its destination because lines 1, 2 and 7 perform the actions described in Stage I of our description above, and we have shown this to be a complete, although inefficient, routing algorithm. The remaining steps are designed to use the routing table to improve the algorithm's performance by reducing the number of hops required.

Lines 4–5 come into play whenever D does not fall within the numeric range of the current node's leaf set and relevant routing table entries are available. The selection of a destination for the next hop involves comparing the hexadecimal digits of D with those of A (the GUID of the current node) from left to right to discover the length, p, of their longest common prefix. This length is then used as a row offset, together with the first non-matching digit of D as a column offset, to access the required element of the routing table. The construction of the table ensures that this element (if not empty) contains the IP address of a node whose GUID has p+1 prefix digits in common with D.

Line 7 is used when D falls outside the numeric range of the leaf set and there isn't a relevant routing table entry. This case is rare; it arises only when nodes have recently failed and the table hasn't yet been updated. The routing algorithm is able to proceed by scanning both the leaf set and the routing table and forwarding M to another node whose GUID has p matching prefix digits but is numerically closer to D. If that node is in L, then we are following the Stage I procedure illustrated in Figure 10.6. If it is in R, then it must be closer to D than any node in L, hence we are improving on Stage I.

Host integration • New nodes use a joining protocol in order to acquire their routing table and leaf set contents and notify other nodes of changes they must make to their tables. First, the new node computes a suitable GUID (typically by applying the SHA-1 hash function to the node's public key), then it makes contact with a nearby Pastry node. (Here we use the term *nearby* to refer to network distance, i.e., a small number of network hops or low transmission delay; see the box below.)

Suppose that the new node's GUID is X and the nearby node it contacts has GUID A. Node X sends a special *join* request message to A, giving X as its destination. A despatches the *join* message via Pastry in the normal way. Pastry will route the *join* message to the existing node whose GUID is numerically closest to X; let us call this destination node Z.

A, Z and all the nodes (B, C,...) through which the *join* message is routed on its way to Z add additional steps to the normal Pastry routing algorithm, which result in the transmission of the contents of the relevant parts of their routing tables and leaf sets to X. X examines them and constructs its own routing table and leaf set from them,

requesting some additional information from other nodes if necessary.

To see how X builds its routing table, note that the first row of the table depends on the value of X's GUID, and to minimize routing distances, the table should be constructed to route messages via neighbouring nodes whenever possible. A is a neighbour of X, so the first row of A's table is a good initial choice for the first row of X's table, X_0. On the other hand, A's table is probably not relevant for the second row, X_1, because X's and A's GUIDs may not share the same first hexadecimal digit. The routing algorithm ensures that X's and B's GUIDs do share the same first digit, though, which implies that the second row of B's routing table, B_1, is a suitable initial value for X_1. Similarly, C_2 is suitable for X_2, and so on.

Furthermore, recalling the properties of leaf sets, note that since Z's GUID is numerically closest to X's, X's leaf set should be similar to Z's. In fact, X's ideal leaf set will differ from Z's by just one member. Z's leaf set is therefore taken as an adequate initial approximation, which will eventually be optimized through interaction with its neighbours as described in the fault tolerance subsection below.

Finally, once X has constructed its leaf set and routing table in the manner outlined above, it sends their contents to all the nodes identified in the leaf set and the routing table and they adjust their own tables to incorporate the new node. The entire task of incorporating a new node into the Pastry infrastructure requires the transmission of $O(\log N)$ messages.

Host failure or departure • Nodes in the Pastry infrastructure may fail or depart without warning. A Pastry node is considered failed when its immediate neighbours (in GUID space) can no longer communicate with it. When this occurs, it is necessary to repair the leaf sets that contain the failed node's GUID.

Nearest neighbour algorithm

The new node should have the address of at least one existing Pastry node, but it might not be nearby. To ensure that nearby nodes are known Pastry includes a 'nearest neighbour' algorithm to find a nearby node by recursively measuring the round-trip delay for a probe message sent periodically to each member of the leaf set of the nearest currently known Pastry node.