Assignment 2

IMPLEMENTING THE PASTRY PEER TO PEER NETWORK
VERSION 0.30

DUE DATE: Wednesday October 14th, 2015 @ 5:00 pm

OBJECTIVE

The objective of this assignment is to build a simple peer to peer network where individual peers have 16-bit identifiers. This assignment has several sub-items associated with it: this relates to constructing the logical overlay and traversing the network efficiently to store and retrieve content. This assignment will be modified to clarify any questions that arise, but the crux of this assignment will not change.

All communications in this assignment are based on **TCP**. The assignment must be implemented in **Java** and the external jar files that you can use are listed towards the end of the assignment. You must develop all functionality yourself. This assignment may be modified to clarify any questions (and the version number incremented), but the crux of the assignment and the distribution of points will not change.

Grading: This assignment will account for **15 points** towards your cumulative course grade. There are several components to this assignment, and the points-breakdown is listed in the remainder of the text. This assignment is to be done individually. The scoring process will involve a one-to-one interview session of approximately 30 minutes where you will demonstrate all the required functionality based on the inputs that will be provided to you. The slots for these interview sessions will be posted a few days prior to the submission deadline.

Generating identifiers **(1 point)**

The first time you create a peer you will rely on local timestamps to generate the 16-bit identifier i.e. the total number of peers in the system can be about 64,000. The system should support both: (1) specification (at the command line) of an identifier during startup of peer, and (2) automatic generation a peer identifier if the user does not provide one during the as a startup parameter. Note the identifiers should be based on hexadecimals; they hexadecimal identifiers harnessed during routing. I have attached my Java code for converting a Hex String into a byte[] and vice-versa in the appendix portion of this assignment. Use this also for printing out the entries in the routing table and so forth. This feature of assigning identifiers statically to nodes will be used during the scoring process.

1 The Discovery Node

There will also be a discovery node in the system that maintains information about the list of peers in the system. Every time a peer joins or exits the system it notifies this discovery node. The registration information includes information about the peer such as:

- Its 16-bit identifier
- The {*host:port*} information (please use TCP for communications)
- A nickname for this node

In the unlikely case that there is a collision in the ID space, the discovery node notifies the peer about this collision at which point the peer will regenerate a new identifier and repeat the process.

The discovery node has been introduced here to simplify the process of discovering the first peer that will be the entry point into the system. The discovery node is ONLY responsible only for

1. Returning **ONE** random node from the set of registered nodes
2. Detect collisions

If the discovery node is used for anything else there is will be a **13 point deduction**. An example of misusing the discovery node is to use it to give a new node information about all nodes in the system: such a misuse will defeat the purpose of this assignment.

Points:

1. Adding and removing entries from the discovery node when a peer either joins or leaves the system. (**1 point**)
2. Returning a random *live* peer's network information when a peer joins the overlay (**1 point**)

2 Protocol for Routing Content in the P2P Network

The primary functionality provided by the DHT is the lookup operation. A *lookup(key)* operation identifies the node with the numerically closest identifier to the key. The routing algorithm for the DHT involves two data structures that assist in routing: (1) Leaf Set and (2) Routing table.

Routing can be done using just the Leaf Set; though the routing solution converges in this case, the solution is inefficient. The best routing solution combines both the Leaf Set and the Routing Table. **Your implementation should support the solution that combines the Leaf Set and the Routing Table**. The description that follows in the remainder of the assignment is based on the Pastry algorithm as described in our recommended text.

2.1 Leaf Set

At a given peer, this data structure is responsible for tracking the $2l$ ***neighbors*** of that peer; l to the right of the peer and l to its left. The DHT ID space can be thought of as being organized as a ring: 0-(2^{16}-1). The neighbors refer to peers whose identifiers are *numerically closest* to the peer in question. So, if there is a set of peers in the system with IDs: 53, 65, 69, 73, 83, 92. Then the Leaf Set at 69, when l =2 , is {53, 65, 73, and 83}.

The simplest routing solution using the Leaf Set involves taking hops (of size >0 and <= l) to reach the destination. At each peer, you will choose a hop that gets you close to the destination; so, in most cases, you will be taking l hops at each peer as you try to get closer to the destination, before using a hop of size 1. For a system with N peers, it will require about N/$2l$ hops to deliver a message using only the Leaf Set (as can be seen, this is very inefficient).

For the purposes of this assignment l=1 i.e. your Leaf Set at each peer involves 2 (numerically) closest peers one with ID greater and one with ID lower than the peer.

2.2 The Routing Table

The routing table maintains information about several peers in the system. All peer identifiers in the DHT are viewed as **hexadecimal** values. The routing table *classifies* peer identifiers based on their hexadecimal **prefixes.** The table has *as many rows* as there are hexadecimal digits in the peer identifier. Thus, in our case, with 16-bit identifiers, there will be 16/4 = 8 rows. Any row in this table contains 15 entries: 1 for each possible value of the *n*th hexadecimal digit. The routing table *excludes entires*/values that correspond to local peer's identifier. As can be seen the routing table (depicted in Figure 1) at a peer, has increased density of coverage for peers with IDs that are numerically closer to its own. The "n" in the cells refers to node handles; each cell contains the IP address and identifier of *one* peer that matches the prefix criteria. Note that some of the cells may be empty since the Routing Table may not be fully populated.

p =	Identifier Prefixes and corresponding node handles n															
0	0 n	1 n	2 n	3 n	4 n	5 n	**6**	7 n	8 n	9 n	A n	B n	C n	D n	E n	F n
1	60 n	61 n	62 n	63 n	64 n	**65**	66 n	67 n	68 n	69 n	6A n	6B n	6C n	6D n	6E n	6F n
2	65 0 n	65 1 n	65 2 n	65 3 n	65 4 n	65 5 n	65 6 n	65 7 n	65 8 n	65 9 n	**65** **A**	65 B n	65 C n	65 D n	65 E n	65 F n
3	65 A0 n	**65** **A1**	65 A2 n	65 A3 n	65 A4 n	65 A5 n	65 A6 n	65 A7 n	65 A8 n	65 A9 n	65 AA n	65 AB n	65 AC n	65 AD n	65 AE n	65 AF n

Figure 1: Example routing table at a node with identifier **65A1**. Each cell contains the IP address and identifier of one peer that matches the prefix criteria. Note that some of the cells may be empty.

The routing solution that uses the Routing Table and the Leaf Set provides the best routing characteristics. Specifically, the algorithm routes requests to any node in *O(log N)* messages.

Let's say that you receive a message from a peer **A** with the intended destination of **D**. You will begin by comparing the hexadecimal representations for peers **A** and **D** from ***left-to-right*** to discover, p, their longest common prefix. You will then consult the routing table at **A** using the p (that we just computed) as the row offset. The first non-matching digit in **D** is used as the column offset, to access the required element in the table. The table construction should ensure that this cell contains the address of the peer that has (p +1) prefixes in common with **D**.

Another check that is performed is if D is in the leaf set of A. If it is, the routing converges and after routing to the destination D no further steps need to be performed.

2.3 Addition of a New Peer and Construction of the Routing Table

New nodes use a joining protocol. This protocol allows a new peer to *acquire* its routing table and leaf set contents; this protocol also includes notifying other nodes of changes that they must make to their tables.
Let's say that a new peer joins the system and this new node's identifier is **X.** This new node contacts it's entry-point node **A** using the Discovery Node**.** Node **X** sends a special join request message to **A** and gives **X** as its destination. This node **A** dispatches the join message via the DHT to an existing node with an identifier that is numerically closest to **X;** we will call this the destination node **Z**.

The join message is routed through the network: **A**, **Z** and intermediate nodes (**B**, **C**, …). This results in the *transmission of relevant parts* of their routing tables and leaf sets to **X.** **X** examines and constructs its own routing table and leaf set from the: entry point peer, intermediate peers, and the destination peer.

In the original protocol, the node that serves as the entry (**A**) has network proximity to the new node **X**. We will be running our assignment in a local cluster, so this technically moot in our case. However, similar to the original algorithm, we will assume that the first row of **A**'s table is a good initial choice for **X** i.e. **A** and **X** will have the same first row. Note: the Routing Tables uses 0-based row indexing i.e. the indices start at 0.

A's table is not relevant for the second row because the GUIDs for **X** and **A** may not share the 1st digit. But the routing algorithm ensures that **X** and **B**'s GUID do share the first hexadecimal digit. So 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.

Since **Z**'s identifier is numerically closest to **X**'s. **X**'s ideal leaf set will differ from **Z**'s *by just one* member. This is **eventually** optimized through interaction with the neighbors.

Once **X** has constructed the its leaf set and routing table, **X** sends its contents to all *nodes* identified in the **leaf set** and the **routing table**. The nodes that receive these updates, *adjust* their own tables to **incorporate** the node.

Points Distribution for Implementing Protocol for Routing of Content:

1. Adding a new node at the right location in the overlay using the correct route **(3 points)**
2. Creating the routing table at the newly added node with the correct entries **(2 points)**
3. Updating the routing table and the leaf set at the nodes that were impacted as a result of this addition. **(2 points)**
4. Migrating data items from peers that are now not the most suitable peers to host the data **(1 point)**

3 Storing data items

You must use complete routing solution (encompassing the routing table and the leaf set) to store data items at the appropriate node. A data item with a key k will be stored at the peer with the *closest* numerical identifier (in the case of ties choose the peer with the higher identifier). The data item that will be given to you will be images. To support this feature you will develop a StoreData program that accepts as input the file that needs to be stored. This StoreData program contacts a random peer (you can contact the Discovery node to retrieve this information). The StoreData program will first compute the 16-bit digest for the image and then use this hash to ***lookup*** the peer where it should be stored: you will be contacting the aforementioned random peer to initiate this lookup; the node that gets back to you will be the node that is most suitable to store your data. The file is then transferred to that suitable peer, which is responsible for storing the file in the /tmp directory of the machine that it is running on.

Points:

1. Identification of peer to store content **(1 point)**.
2. Using the routing tables and the leaf set to take the correct route to reach the targeted peer **(3 points)**.
3. Storing the file in the /tmp directory of the target peer **(1 point)**

4 Diagnostics

To make sure that things are progressing correctly this assignment requires that several diagnostic information be printed on the console. These include:

1. The routing table and the leaf set at a node
2. The list of files managed by the node: This would be stored in the /tmp directory of the machine on which the peer is running
3. Print out a message every time you route a query: This message should indicate the hop number that it corresponds to in the routing path. So, if a lookup() operation has bounced off of 3 nodes, and is now received at a node … it should print a hop count of 4.
    a. We will use this information to reconstruct the path which the lookup/successor operations took.
    b. In the absence of this hop information, all that would need to be done is to sort the peer ids appropriately to simulate the correct path.

Lack of diagnostic information for checking the correctness of your routing paths will result in a **6 point deduction.**

5 Programs that you are responsible for developing

In summary you are responsible for developing 3 distinct programs

1. Discovery Node
2. Peer Node with a configurable port number that prints out several diagnostic information
3. StoreData which is responsible for ensuring the storage of the content on the correct peer in the system

6 Third-party libraries and restrictions:

You are not allowed to download *any* other code from *anywhere* on the Internet. You are also not allowed to use RPC or distributed object frameworks to develop this functionality (there is a **15 point deduction** for this). You should not build GUIs for this application; in the context of this assignment GUI-building is an auxiliary path (there is a **15 point deduction** for building a GUI). You can discuss the project with your peers at the architectural level, but the project implementation is an individual effort.

7 Testing Scenario

You will be asked to launch between 10-20 processes possibly on different machines. The port number on which your peer runs and listens to for communications should be configurable.

Submission deadline:

Please submit the source codes for your project by the 5:00 pm on the due date. This should be mailed to cs555@cs.colostate.edu. There is a **1 point deduction** for mailing it to the instructor's e-mail account. We will rely on the honor system: please do not make any modifications to the codebase after the submission deadline has elapsed.

Change History

Version	Date	Change
0.30	9/5/2015	First release of the assignment

Appendix A

```
/**
 * This method converts a set of bytes into a Hexadecimal representation.
 *
 * @param buf
 * @return
 */
 public String convertBytesToHex(byte[] buf) {
      StringBuffer strBuf = new StringBuffer();

      for (int i = 0; i < buf.length; i++) {
            int byteValue = (int) buf[i] & 0xff;
            if (byteValue <= 15) {
                  strBuf.append("0");
            }
            strBuf.append(Integer.toString(byteValue, 16));
      }
      return strBuf.toString();
}


/**
 * This method converts a specified hexadecimal String into a set of bytes.
 *
 * @param hexString
 * @return
 */
public byte[] convertHexToBytes(String hexString) {
      int size = hexString.length();
      byte[] buf = new byte[size / 2];

      int j = 0;
      for (int i = 0; i < size; i++) {
            String a = hexString.substring(i, i + 2);
            int valA = Integer.parseInt(a, 16);

            i++;

            buf[j] = (byte) valA;
            j++;
      }

      return buf;
}
```