CS 677 Lab: Design Document

1. **Design Considerations –**
    a. *Peer Structure:*

    Each peer will have a peer-id, host address and role which states whether it is buyer or seller. The details about each peer are stored in the config file (FileName: config). We use a fully connected/ring/mesh topology for our peers. The number of peers is equal to the size of the config file.

    b. *Choice of Communication:*

    We are using RMI for communications in Java. RMI is Remote Method Invocation. It allows an object residing in one JVM to access/invoke an object in another JVM. We used RMIs registry interface to map names to remote objects. In this we use a server that registers its remote objects that can be looked up by others using its name.
    If the default port for RMI(1099) is busy then we provide alternate ports for the registry creation.

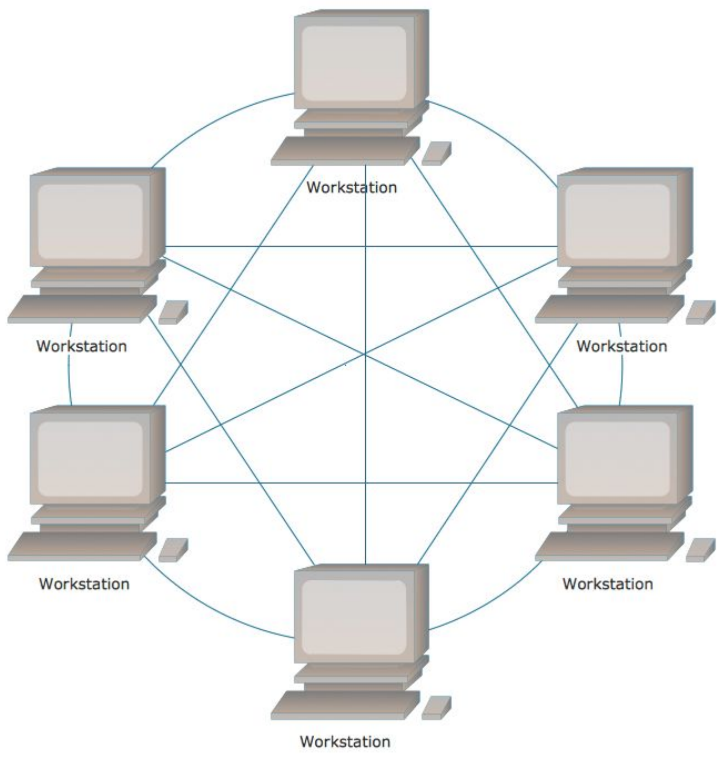
    c. *Neighbors File:*

    The details of peers being neighbors of each other is stored in this file. This file is parsed each time we try to configure neighboring peers and establish a connection amongst each other.

2. **Implementation Details –**

    The broad idea of the implementation revolves around using data structures to store buyer and seller configurations like products they are selling/buying with their quantities.
    For each seller, the product he is selling and its quantity is decided randomly. Also for each buyer the product he is looking up for is done using random assignment.
    We start with randomly initialising peers as buyers and sellers and starting lookup using query flooding. In this setting peers are connected over an overlay network. It means if a connection/path exists from one peer to another, it is a part of this overlay network. In this network the peers act as nodes and the connection between these nodes are the edges, thus resulting in a graph-like structure as shown below.

CS 677 Lab: Design Document



*each workstation acting as a peer(fully connected topology)

a. *Lookup_product (`lookupProduct()`) -*

We implement our lookup functionality in Peer class. In this we start by creating an iterator over the neighbors of a currentPeer. Then we create a LinkedList to maintain a path from buyer ID to the seller ID. This list helps in tracing back to the identify which buyer initiated the lookup. If we don't find the seller in the immediate neighbourhood, then this list helps us in locating the seller in the neighbours of neighbours. The current hop count is the size of this LinkedList.

b. Reply -

The reply from the seller is represented in the ReplyTask class in our implementation. In this we first collect the request from the requestItem map and make sure buy is called. Also we have a boolean success that makes sure that buy is successful because there are multiple buyers buying at the same time. This entire block is synchronized using thread pool so that if there are multiple buyer threads they have to wait until the critical section is free. This will make sure that response time is improved.

c. *Buy -*

The buy function is implemented in the BuyTask class in our implementation. In this synchronized block we first check if the inventory is same as the item that buyer is requesting, if this is not the case it returns false. If the inventory is available then its value is reduced to issue the selling. If the product is over then we initialise the inventory again. If the seller is found, we backtrack over the LinkedList by popping out the ID's pushed so that we get the ID of the buyer. This case can also happen if the size of the linkedList is equal to the maxHopCount.

d. *Thread Pool for multiple buyers and sellers:*

We use Thread Pool in order to obtain multiple concurrent buying and selling. A thread pool reuses previously created threads to execute current tasks and offers a solution to the problem of thread cycle overhead and resource thrashing. In our implementation we initialise the thread pool to 5, but it can be changed to any number in order to maintain multiple buyers and sellers.

3. Design Advantages -

Our code supports a wide variety of concurrency -

1. Any peer in our design can be made a buyer or seller. Each of them support both the functionalities.
2. Our design works for any kind of combination of buyers and sellers like 1 seller 1 buyer, 1 seller 2 buyer, 2 buyer 1 seller, all buyers and all sellers, only buyer and only seller.
3. Our design also supports fully connected ring and mesh topologies.
4. Our design is very easy to operate because most of the work is automated and can be configured very easily.