This is the design document for our P2P Bazaar system. We have chosen to implement the system in Java, and used its RMI package for remote methods. First we describe how to run the system. Then we go on to describe the system design and flow. Following that, we describe the remote interface, and the methods it exposes.

How to run the system

1. To run the system, the configuration needs to be specified in the config.properties file in the src directory. The configurable parameters are number of nodes, hop count, and starting stock for a seller every time it chooses a product to sell. Apart from these parameters, the IP address and port number of every peer in the system must also be specified in the format :
   <nodeId> : <IP:port>
   The node Ids should be integers ranging from 1 to N, where N is the number of nodes in the network.
2. Then, the source files need to be compiled by running javac *.java in the src directory.
3. Once the source files are compiled, each peer in the system needs to be started in a separate terminal, either on the same or different machine, by running
   java Peer <nodeId>
   The nodeId is the nodeId as specified in the config file.

System Design and Flow

We have designed the system to follow a ring topology, where each peer is connected to two neighboring peers.

Each peer, upon startup, first obtains the system configuration from the config file, and validates it. The number of nodes specified has to be >= 3, the hop count should be > 0, and the starting stock for a seller should be > 0. Then, it checks whether the nodeId given as input is in the correct range, which is [1,N]. If any of these checks are not passed, the program terminates with an error message.

Upon passing the above checks, each peer creates a registry on the IP and port specified in the config file against its nodeId, and binds its interface to the registry.

Post binding its interface, it then tries to establish connections with its neighbors. Specifically, it tries to obtain references to its neighbors' interfaces by locating the neighbors' registry and looking up the registry for the interface bound to it. A peer with nodeId 1 has as neighbors nodes 2 and N. A peer with nodeId N has as neighbors nodes (N-1) and 1. A peer with a nodeId between 1 and N, has as neighbors nodes (nodeId-1) and (nodeId+1). See the picture below for reference.

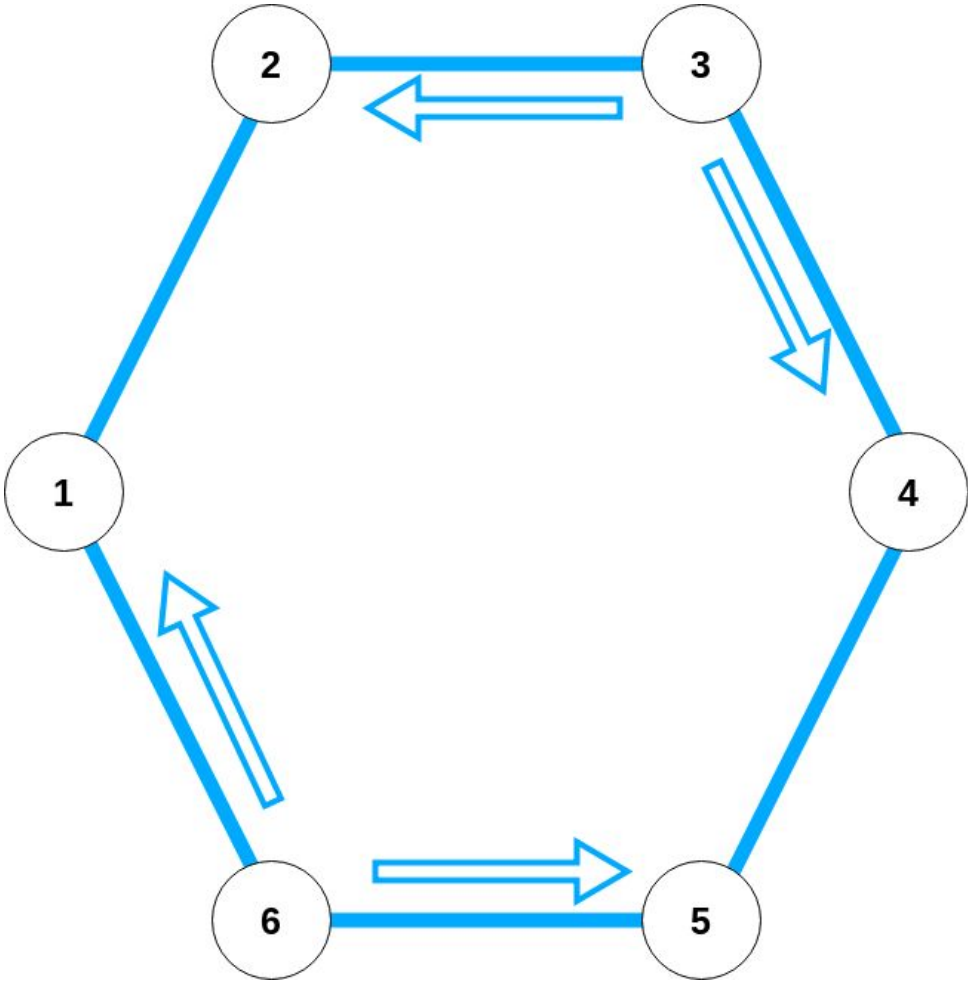

Fig. Network topology. The connections shown are for nodes 3 and 6 only. Likewise, every node will connect to its two neighbors

Once the references are obtained, the peer randomly chooses a buyer or seller role for itself. Then, depending on whether the peer is a buyer or a seller, execution proceeds.

Buyer : If a peer is a buyer, it forever tries to buy products. Specifically, it randomly chooses a product to buy, and does a lookup for the product. In return it gets a list of nodeIds which are selling that product. It then randomly chooses a peer out of the list, and tries to connect to it. If it is successfully able to connect and obtain a reference to its interface, it tries to buy from the chosen peer. If the buy completes successfully, it then waits for a random number of seconds between 1 and 10. And then chooses a new product to buy, and the cycle repeats. If the buy doesn't succeed, the peer removes that nodeId from the list, and then randomly chooses another one to buy from. Like so, it keeps trying to buy from the peers whose IDs it got back as a reply to the lookup, until it is either successfully able to complete a buy, or the replies list is exhausted. If the replies list is exhausted, it randomly chooses another product, and the cycle repeats.

Seller : If a peer is a seller, it forever tries to sell. Specifically, it chooses a random product to sell, and sets its stock to the starting stock specified in the config file. Once the stock gets over, a new product is randomly chosen to sell, and the peer is restocked with it. This happens in the seller's buy method. Every time the buy method is invoked, it checks at the end if the stock is over.

Interface and its methods

The interface exposes the following methods :

Buy :
The signature of the buy method is as follows.

boolean buy(int nodeId, int productId)

It accepts a nodeId and productId as input. nodeId is the id of the peer calling the buy method, and the productId is the id of the product it is looking to buy. Upon receiving a buy request, a peer first checks if it is a seller and whether the productId in the input is the product it is selling. If it isn't, it returns false. Then, it checks if the current stock is > 0. If it is, it prints to console that it sold the specified item to the requesting peer, and decrements the seller's stock by 1. After decrementing, it checks whether the stock is now zero. If it is, it randomly chooses another product for the seller to sell, restocks the seller, and returns true.

We have made buy a synchronized method, so that if a peer (peer A) is executing a buy method invoked by some peer (peer B), no other peer (peer C) can simultaneously execute the buy method on this peer (peer A). This is to avoid race conditions between peers B and C.

Lookup
The signature of the lookup method is as follows.

List<Integer> lookup(int nodeId,int productId,int hopcount)

The lookup method initially checks if the hop count is 0. If it is, it then checks if it is a buyer. If it is a buyer, it returns an empty list. If it is a seller, it checks whether it is selling the requested product. If yes, then it returns a list with its own ID in it. If it is not selling the requested product, it returns an empty list.

If the hop count is not 0, then the peer first calls lookup on its neighbor, other than the one which called lookup on the current peer. It gets a list as a reply. Then, it checks whether it itself is a seller, and is selling the product requested. If it is, it appends its own ID to the list it received as a reply from its neighbor, else it does not append. Finally, it returns the list to the caller.

getNodeId

This method just returns the nodeId of the peer it is called on.