

Distributed System Lab 1 Design Document

Chuu-Hsiang Hung, Chuanpu Luo

1. Overall Program Design

The overall program design UML is shown below. In each Terminal, script Main reads configuration information from test file and generates a Node instance to run. Based on our design, a Node instance is similar to a process, it has its own IP address, its own Message Queue to handle and its own neighbor IP address map. Each Node runs a thread to provide an RMI function named “send” for other Nodes, “send” function receives Message from other Nodes and add it into its own Message Queue. Also, each Node runs another thread to check its Message Queue all the time, handle the Message based on different Message type (LOOKUP, REPLY and BUY) and different Peer type (Buyer, Seller, BuyerAndSeller). In Part 2 we will describe how this program works in detail.

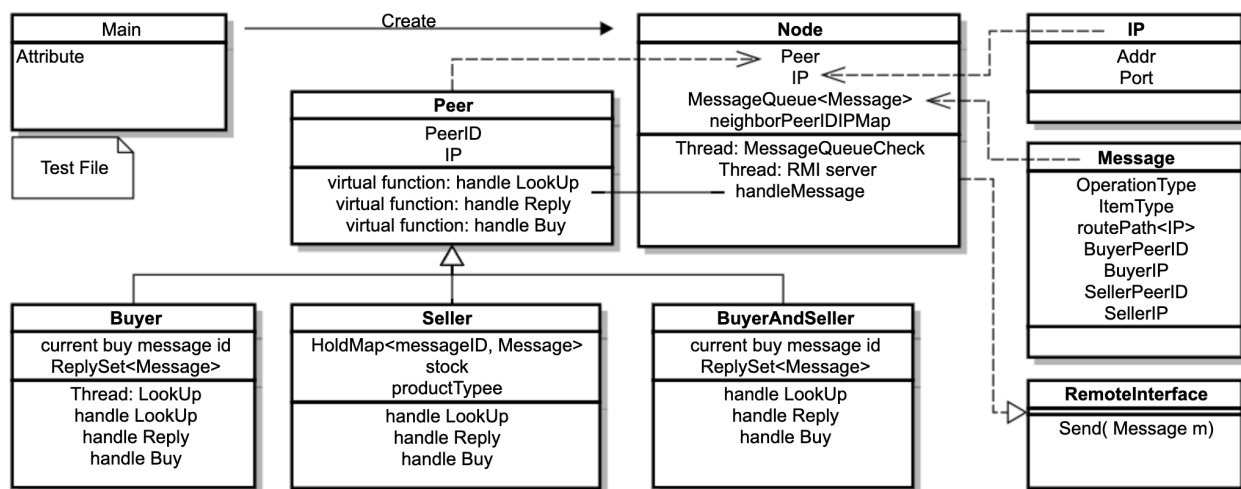


Figure 1. Program Design UML

2. How it works

a. Blocking Message Queue

For each node, we run two threads: **RMIserverThread** and **MessageQueueCheckThread**. The former binds the server instance to the rmi registry and the latter consumes the messages. Each node maintains a message queue of **BlockingQueue** class. When remote method "send" is invoked, the message is added to the remote node's message queue. The message is then consumed by invoking "take" method from **LinkedBlockingQueue** class. It blocks the other threads until the message queue is not empty so we can ensure that we can always get message to handle from queue.

b. Trade communication between peers

After a specific time interval (set up in **Node.java**), buyer will randomly pick a product to buy and send "LOOKUP" Message to all its neighbors. Node who receives "LOOKUP" Message will decide if it can provide the product or not. If a Node can provide the

product, it will send "REPLY" Message in reverse direction back to the buyer. If a Node cannot provide the product, it will flood the "LOOKUP" Message to its neighbors. Once the buyer receives multiple "REPLY" message, Buyer will randomly pick a Seller, and send "BUY" Message directly to the Seller. In this way, a transaction is finished.

c. Mechanism to prevent duplicate buying

When a Buyer tends to buy a product, it will generate and spread "LOOKUP" Message to its neighbors. One of the mechanisms that we use to prevent duplicate buying is, giving these "LOOKUP" Message the same Message ID. When Seller replies "REPLY" Message, Seller will not change the same Message ID. In this way, even if a Buyer receives several "REPLY" Message, it knows these Messages are from the same transaction.

d. Mechanism to "Lock" product while selling

When a Seller receives "LOOKUP" Message, it will reply "REPLY" Message to the Buyer, to tell Buyer that it has the product. But the Seller would not decrease its product stock, because it does not know if the Buyer would pick it and send "BUY" Message to it. In this way, the Seller need to hold the product for this Buyer. In order to do so, we create a Message set to save the "LOOKUP" Message named Hold Set. There are products available only when the size of Hold Set is smaller than the size of stock. Also, for those Sellers who hold products for Buyers but not picked by Buyers, we set up a maximum time for Sellers to hold these products. If they don't receive "BUY" Message within the maximum hold time, Sellers would drop "LOOKUP" Message and stop holding products for Buyers.

3. Design Tradeoffs

In this lab assignment, we keep a Message Queue in each Node and use RMI for nodes to communicate with each other. The only RMI function we designed in this lab is "send" function, to receive Message from other Nodes and add it into Message Queue. Actually, there is no necessary to keep a Message Queue if we use RMI to communicate between Nodes: one Node could call functions like "HandleLookUp", "HandleReply", "HandleBuy" directly from other Nodes. However, we think it's better to keep a Message Queue for each Node, and each Node handle these Messages by its own. We have to admit that keeping a Message Queue is not as efficient as call RMI function directly, but keeping a Message Queue makes code more structured and helps to prevent concurrency problems.

4. Possible Improvements and Extensions

In this lab, we assume that the network is static, it means that there is no new Node to add in and no existing Node to quit. However, in real world distributed system network, Nodes add in and quit at every moment. So designing rules allowing Nodes to add and quit is one of the most important extensions.

Another extension is adding time stamp in each transaction. Although there is no global time for the whole system, we should not use local time of any single machine to stamp the transaction because it could be wrong.

5. How to Run

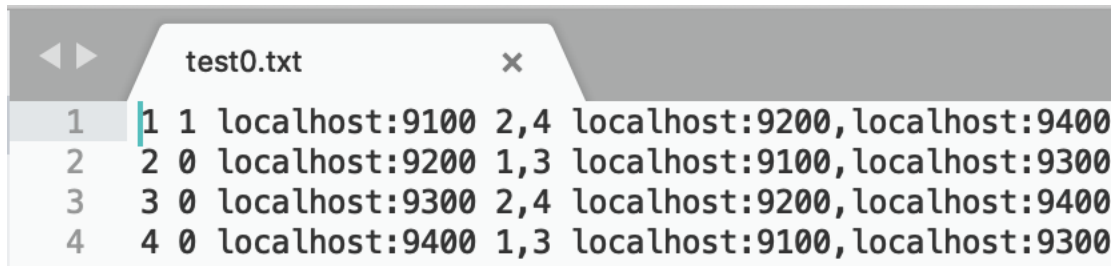
Compile Source Code in Linux:

```
$ cd lab1-1-hung-luo/src
```

```
$ make clean
```

```
$ make
```

Run Nodes:



1	1	1	localhost:9100	2,4	localhost:9200,localhost:9400
2	2	0	localhost:9200	1,3	localhost:9100,localhost:9300
3	3	0	localhost:9300	2,4	localhost:9200,localhost:9400
4	4	0	localhost:9400	1,3	localhost:9100,localhost:9300

Here is an example of test file. There are 4 Nodes configuration in test0.txt. Each line represents one Node. For the first line, from left to right: Node Peer ID, Node Type (Buyer 0, Seller 1, BuyerAndSeller 2), IP, Neighbor Peer ID, Neighbor IP.

In order to run this test file. We have to open 4 terminals and run one Node in one terminal. Command line should be: `java Main [Test File Directory] [Node Peer ID]`

For example, if we would like to run test0.txt, we have to type in each command in one terminal:

```
$ java Main ../test/test0.txt 1
```

```
$ java Main ../test/test0.txt 2
```

```
$ java Main ../test/test0.txt 3
```

```
$ java Main ../test/test0.txt 4
```