

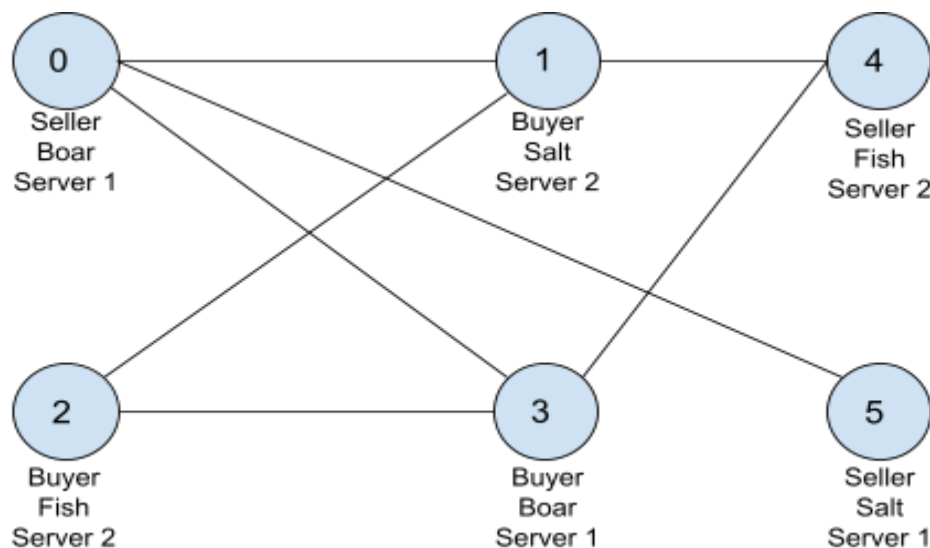
# DESIGN SPECIFICATION

This project implements a peer-to-peer network with multiple peers depending upon the values of N and K which can be specified in the config.txt file as mentioned in the project requirements.

## Network Initialization

Based upon the value of N and K, a random P2P network is initialized. For every peer in N, K peers are randomly selected and are initialized as the neighbors of that peer. Peers can be deployed in different servers but may also be deployed on the same server. Roles are randomly assigned to every peer (Seller or buyer).

For example, if we set the value of N and k as 6 and 3 respectively, the following network can be generated after random initialization.



## Network Execution

A buyer sends a lookup request to its neighbors which is propagated further in the network through the flooding procedure. After an appropriate seller is found, the reply is sent back from the seller to the buyer through the same path. When the request reaches back to the buyer peer, the buyer adds the specific seller to its seller list. Similarly, all the sellers who respond to the request of a buyer get added to the buyer's seller list. After all the flood requests come back, the buyer randomly chooses a seller and proceeds for the connection. Once a connection is established between the buyer and the seller, the buyer sends a buy request to the seller and gets a response if it was able to successfully buy the item. If the seller chosen does not have

the item anymore(i.e. the item was sold between request and reply) then the next seller is chosen.

The seller starts with selling m=5 (default value) quantities of an item. Once the item stock finishes, it restocks a random item from the item list {'boar', 'salt', 'fish'}.

Similarly, the buyer starts with an item to buy, and when it buys it successfully, it tries to buy another random item from the item list {'boar', 'salt', 'fish'}.

## Implementation Level logic

At the code level, we instantiate and deal with peers through the 'Node' class. Each peer is instantiated with a peer ID, role, port it listens on, its neighbors, item it sells or buys and the count of it.

### Interfaces of a Peer:

#### Private:

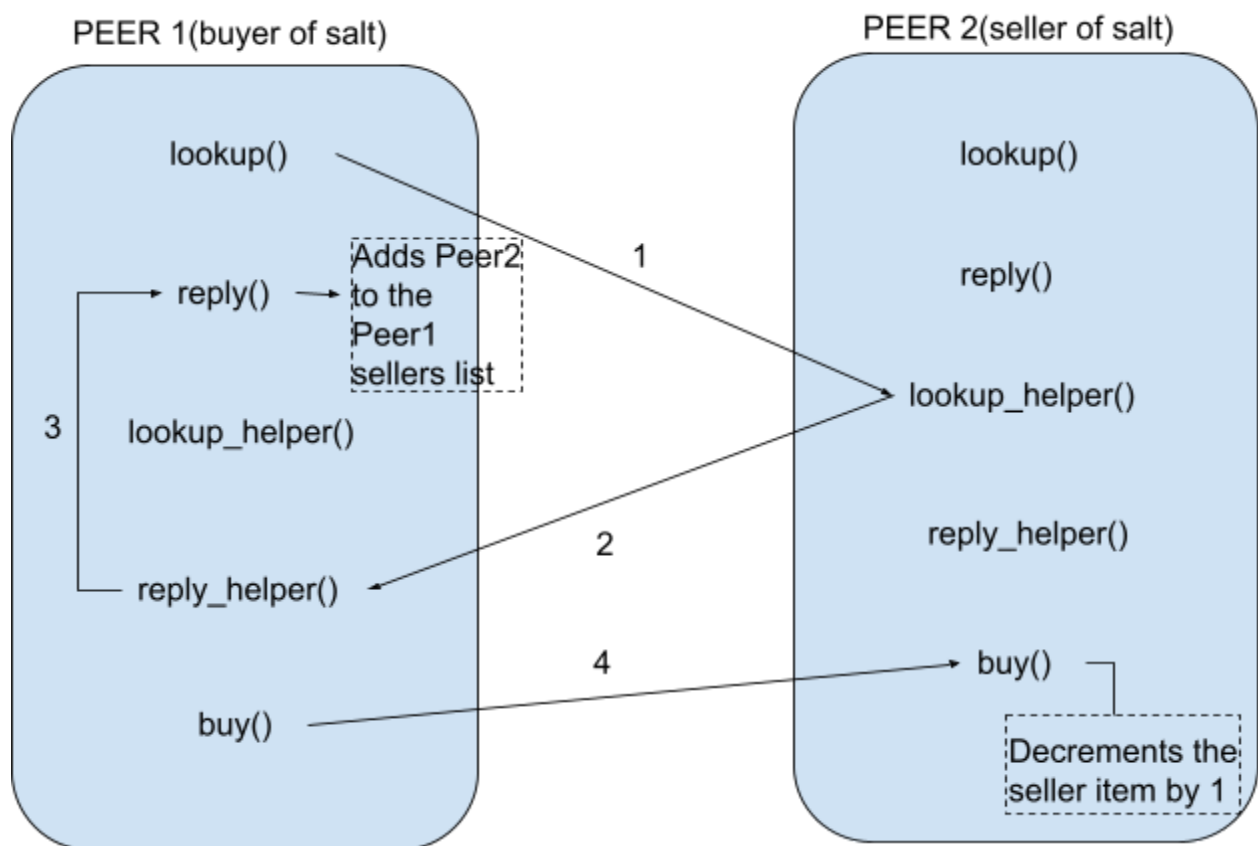
- **lookup(productname, hopcount):**
  1. This interface is called by the buyer to look up the items in the neighboring peers.
  2. This is achieved by calling the lookup\_helper function of all the neighboring peers. While calling, it passes product-name, hop-count, and an array having only the buyers ID as the arguments.
- **reply(buyerID, sellerID):**
  1. This interface is called in case the back propagated request reaches back to the buyer.
  2. Upon being called, it adds the specific sellerID to the buyer's list of all the sellers who responded affirmatively for the lookup item.

#### Public:

- **lookup\_helper(product-name, hop-count, <array of traversed peers>):**
  1. This is a public helper function for private interface lookup.
  2. This interface basically checks if the current peer is seller or buyer and acts accordingly.
  3. If the peer is a buyer, the interface will forward the incoming request to its other neighboring peers(flooding).
  4. If the peer is a seller, the interface checks if the item requested matches with the item it is selling. If yes, it responds back to the peer from where the request came by calling the reply\_helper function of the last traversed peer. Otherwise, it forwards the request to other peers.
- **reply\_helper(<array of traversed peers>, sellerId):**
  1. This interface is called by lookup\_helper of the next peer when the peer identifies itself as a seller of the required item.

2. This in turn calls the previous node lookup\_helper function until it reaches the buyer.
  3. Once the backpropagation request reaches the buyer, it calls the reply function of the same node to add the seller to the buyer's seller list.
  4. When the peer(seller) realizes that the inventory is finished, it restocks the items according to different scenarios.
- **buy(sellerId):**
    1. If the peer is a buyer, it calls the buy interface of the specific seller to buy a particular item.
    2. If the peer is a seller, it checks if the item is still present in its stock. If yes, it confirms the transaction and returns. Otherwise, it sends a negative response to the buyer that the item is no longer available.

Taking an example of a 2 peer network, where Peer 1 is a buyer of salt and Peer 2 has 5 items of salt to sell, following is the flow that will occur once the buyer sends the lookup request. Upon a successful buy, the buyer will choose a random item to buy and initiate the lookup request again.



## How it works

- A config.txt file is used to configure the number of peers (N) in the desired network as well as the maximum number of directly connected peers of each peer(k) with  $k < N$ . The Node ID (peer ID), Port Number as well as IP address of each peer can be configured manually.
- The Initializer file reads this information from the config.txt file and generates a completely random network and saves it in the network.txt file. Each peer is randomly assigned a buyer or seller role. Each peer is randomly assigned an item to buy/sell. The initial count of items for every seller is by default set to 5.
- Each peer is initialized with its peer ID and the peer automatically reads the port, IP, role and gets the list of neighbor peers from the config.txt and network.txt files.
- Upon initialization, the buyer peers will propagate lookup requests to their neighbors while the seller peers will receive incoming requests and either forward them to their neighbors or reply back along the path of the incoming request depending on whether it has the required item or not. Sellers will pick a random item to sell and upon selling all units of the item, will restock another random item. The same goes for buyers as well, where, after a successful transaction the new desired item will be initialized randomly.

## Future Improvements

Following are the ways that can be considered for future improvement of the P2P network architecture that we have defined:

- 1) Peers can be made to work on multiple machines located in multiple subnets/networks. This would require providing appropriate permissions such as firewall, making sure the machines are accepting connections on required ports, etc
- 2) Multithreading can be introduced when each peer either generates lookup requests to its peers or forwards a lookup request to its peers. This way each peer can call its neighbors parallelly instead of doing it sequentially, leading to overall improvements in lookup search time.
- 3) Overall performance and stability improvements along with code optimization.

## Steps for Testing

### System requirement

Local machine (Windows)  
Ec2 servers (Linux)  
Java (Both on the local machine and the ec2 servers)

### Source File Descriptions

**src/Hello.java:** This is an interface file that defines the public function of a peer object.

**src/Node.java:** This is a Network Peer file which implements the interface as well as local functions related to the peer. Running the command `java Node <peer_id>` instantiates a specific peer.

**src/Network.java:** This file defines the P2P Network.

**src/Initializer.java:** This file reads the config.txt file and creates a network and put the network in the file src/network.txt.

**src/config.txt:** The first line in this file contains the number of peers in the network (N) and the maximum number of neighbours of each peer (k). Each subsequent line contains the peer ID, port and IP of each peer (For Example: '1 8011 127.0.0.1' if peer 1 is running on a local machine).

**src/network.txt:** Each line in this file contains information of a specific peer. It contains the Peer ID, a list of neighboring peers, role of the peer, the item peer wants to buy or sell, and the count of the item it starts with (if seller). (For Example: 0 [3,5] buyer Salt).

## Instructions

### To run the network locally

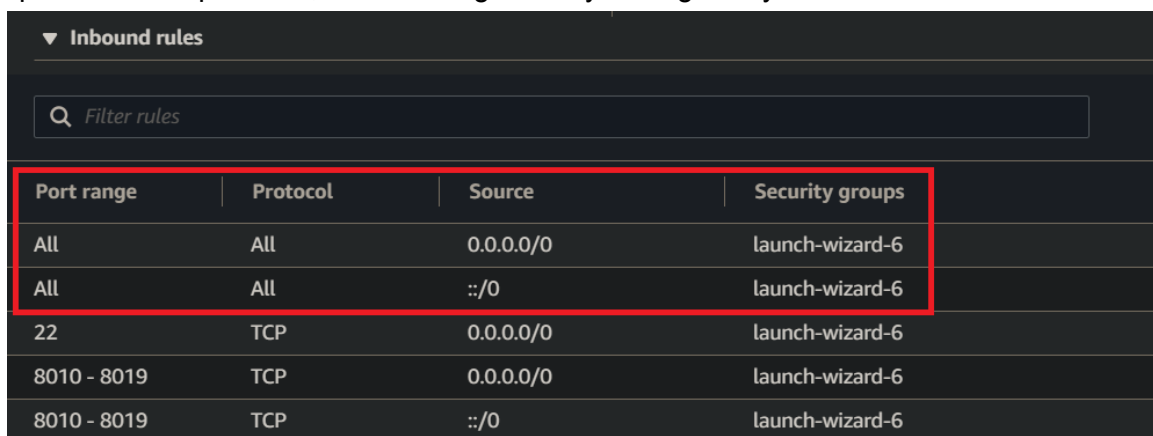
1. Go to directory src which contains the source and config files.
2. Modify the file src/config.txt as desired.
3. Modify the file src/network.txt if you need a specific network of peers. Otherwise this will be generated randomly.
4. Run the following command in the src directory: `python runme.py`.

#### Use-Cases:

- a. To run the network for T seconds: `'python runme.py -t T'`
  - b. To provide a custom network: `'python runme.py -n <network.txt file path>'`
  - c. To kill existing processes: `'python runme.py -k yes'`
- (Important: See the section Usage of script runme.py for more details.)

### To run in EC2 servers present in same network

1. Create EC2 instances with key pair value, and get the private .pem file (We used Amazon Linux 2 AMI 2.0.20210303.0 x86\_64 HVM gp2 (ami-0fc61db8544a617ed) for testing).
2. Edit the security group to ensure that the ports required by the peers to communicate are open. In our experience, the following security setting always works.



▼ Inbound rules			
Q Filter rules			
Port range	Protocol	Source	Security groups
All	All	0.0.0.0/0	launch-wizard-6
All	All	::/0	launch-wizard-6
22	TCP	0.0.0.0/0	launch-wizard-6
8010 - 8019	TCP	0.0.0.0/0	launch-wizard-6
8010 - 8019	TCP	::/0	launch-wizard-6

But you can also try other security settings as long as they provide connection between the ports in different machines.

3. Set up password-less ssh from the local machine to the ec2 servers by running the following command from the local terminal: `ssh -i <pem_file_path> ec2-user@<ec2_public_ip>` with the private pem file and the public IP address of the EC2 instance that has been set up. (This will add the ec2 server to the known\_host file so that you can ssh from the script without the need of a password).
4. Run the following commands on the EC2 instance to install java libraries:  
`sudo yum install java-1.8.0-openjdk`  
`sudo yum install java-devel`
5. Now, on your local machine, go to the directory src which contains the source and config files.
6. Modify the file src/config.txt as desired.
7. Modify the file src/network.txt if you need a specific network of peers. Otherwise this will be generated randomly when you execute the python script.
8. Run the following command in src directory: `python runme.py -pem <pem file used to ssh to all the machines>`  
Use-Cases:
  - a. To run the network for T seconds: `'python runme.py -pem <per_file_location> -t T'`
  - b. To provide a custom network: `'python runme.py -pem <pem_file_location> -n <network.txt_file_location>'`
  - c. To kill existing processes on remote: `'python runme.py -pem <per_file_location> -k yes'`
9. To check the logs of peers specific to each machine, go to the folder `~/P2Pnetwork/docs/` in the EC2 servers and look for the log file `peer<i>.log`.

### Usage of script runme.py

usage: `runme.py [-h] [-pem PEMFILE] [-n NETWORK] [-k KILL] [-t TIME]`

optional arguments: `-h, --help` show this help message and exit

`-pem PEMFILE, --pemfile PEMFILE` Provide RSA Private key file

`-n NETWORK, --network NETWORK` Provide the network.txt file to use a custom P2P network.

`-k KILL, --kill KILL` Provide yes to kill the processes on the ports specified in the config file.

`-t TIME, --time TIME` Provide the time in seconds for which you want to run the processes.