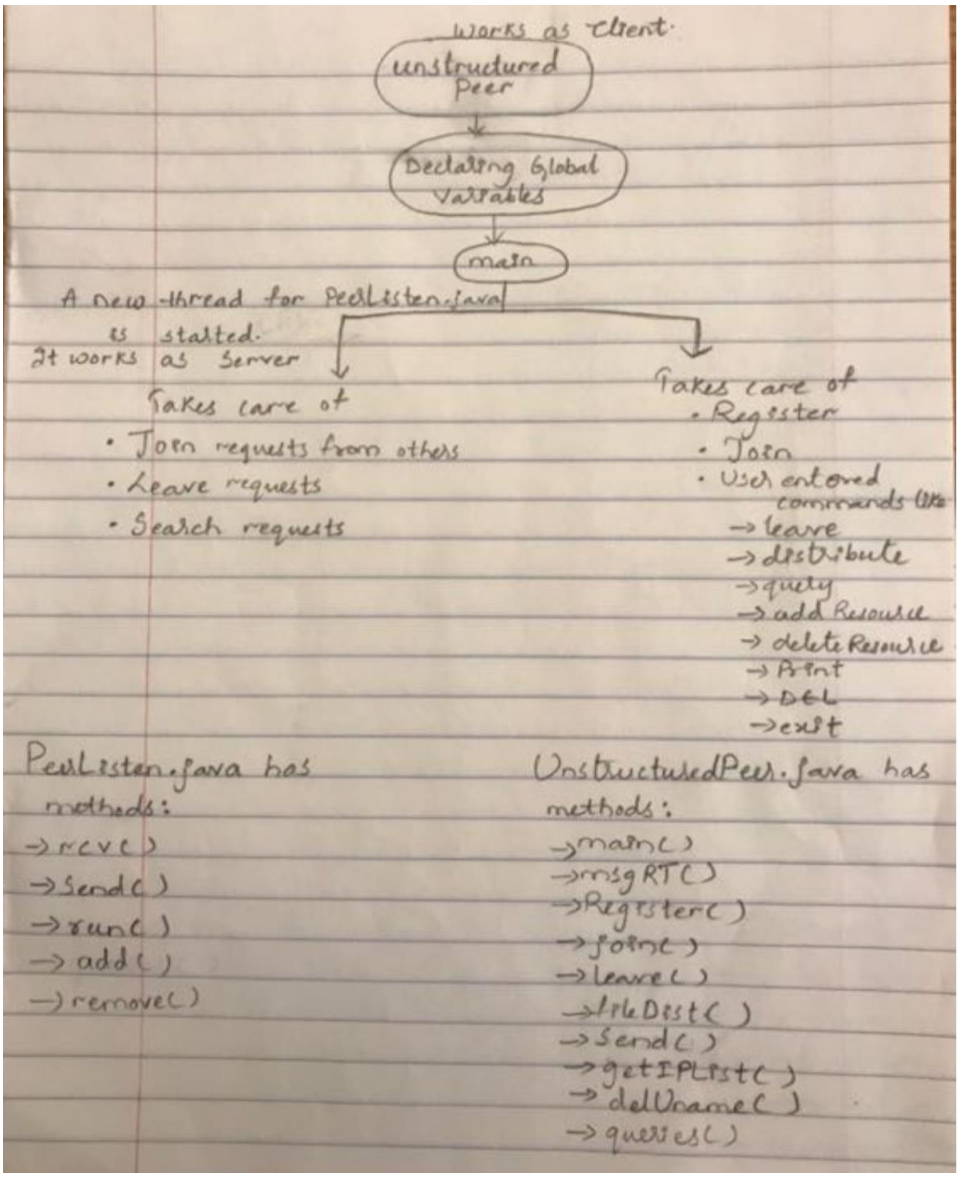
Suchindra Mani Gopavaram
Ravi Teja Jagarlapudi

Lab – 04

Sharing Contents in an Unstructured Peer to Peer Network

The main aim of this project is to develop a simple overlay-based unstructured P2P network which allows all the nodes that are connected to it to share contents/resources among them. The flow of our program written for this project is as follows:



The code is divided into two parts.

- unstructuredPeer.java
- peerListen.java

Suchindra Mani Gopavaram
Ravi Teja Jagarlapudi

One-part acts like a client and the other part acts like a server. The unstructuredPeer.java acts like client and the peerListen.java acts like server.

unstructuredPeer:

The unstructuredPeer is responsible for the following responsibilities:

REG: It sends a registration message according to the given syntax which includes the username for the network to the bootstrap server and gets itself registered in the bootstrap server.

REGOK: It receives the REGOK reply message sent from bootstrap server and checks for the socket addresses in this message, if sent any by bootstrap server. If bootstrap server did send any socket addresses, it will store them.

New Thread: After registering in the bootstrap server, a new thread is started to make the node run in the server mode.

JOIN: The join request message is sent to all the socket addresses of the nodes given by bootstrap server immediately after registration and starting a new thread. So, the first node being registered into the bootstrap server will get zero node IP's from bootstrap. The second node will receive the first node's IP, third node will receive first two node's IP's and the fourth node will receive the first three nodes IP's. from the fifth node randomly any three nodes will be returned by the bootstrap server.

JOINOK: The corresponding JOIN message reply, JOINOK is received by the unstructuredPeer. If the response of the JOINOK message indicates error, then the respective socket address will not be added to the routing table. With the end of this step the network topology will be established, and all the nodes will be communicating with UDP protocols.

User Entry Commands: Now at this stage, with the network topology established, unstructuredPeer waits for any commands entered by the user. The usage of such commands are as follows:

→ add <Resource name>:	Adds resource to the node.
→ delete <Resource name>:	Deletes resource from the node.
→ leave:	Leaves the network.
→ DEL UNAME <username>:	Deletes the network <username> from Bootstrap Server
→ print routing table:	Prints routing table.
→ print routing table size:	Prints the size of routing table.
→ print resources:	Prints resources in this node.
→ distribute <resources per node>:	Distributes the resources.txt contents to all the nodes in the network.
→ query <Filename>	Searches for this file name.
→ exit:	Exits the program.

LEAVE: When user enters a "leave" command then leave method is called. This method makes sure that the node informs all the nodes in it's routing table and the bootstrap server before it leaves the network.

Suchindra Mani Gopavaram
Ravi Teja Jagarlapudi

Distribute: This user entry command when given in any one node of the network distributes the resources to all the nodes in the network.

Add Resource: This command calls the add method in peerListen.java which provides the login in adding a resource to a node. If the resource is already present in the node then the resource will not be added.

Remove Resource: This command calls the remove method in peerListen.java which provides the login in removing a resource to a node If the resource is not present in the node.

Print: This print command has three parts.

- **Print routing table:** This command prints the routing table of the node.
- **Print routing table size:** This command prints the size of the routing table of the node.
- **Print resources:** This command prints the resources in the node.

DEL: This command calls the delUname method which deletes the network in the bootstrap server using its user name.

Queries: The queries method will generate the desired number of queries with the specified zipf's distribution. So, once the first query is generated the node checks its own resources and if found will not forward the search message. If not found will send the search message to all the nodes in its routing table. Once the entire resource or part of the resource is found in a node then it sends the found resource message to the query requested node. If the query is generated for a message which is not present in the network, then the search message will be flooded in the network and after time out the packet dies.

peerListen:

The peerListen acts as a server and will always be in the continuous listening mode. The peerListen is responsible for the following responsibilities:

Run(): The run method in the peerListen class will be listen for incoming messages. Once it catches the message it performs operations appropriate to it.

> JOIN: unlike JOIN in unstructuredPeer which sends the JOIN message, peerListen receives the JOIN message from other nodes and adds that node to the current node's routing table.
>
> LEAVE: When LEAVE message is received from a node, then that node will be removed from the routing table.
>
> SER: When SEARCH message is received from a node, it checks its own resources and if found sends a response else will forward the search message to all the nodes in the routing table.

Rcv (): This method is responsible for receiving the messages from the other nodes and keep the server in the listening mode.

Send (): This method is responsible for responding to the messages received by the rcv() method.

Add (): Using this method the resources will be added to the node.

Remove (): Using this method the resources will be removed from the node.