**Design Document**

**Unstructured Peer toPpeer Network with Cacheing**

**Introduction:**

Unstructured peer to peer network is the network in which each node is connected to a peer node in the network. This does not make use of a central server. In order to maintain the list of clients in the network, a bootstrap server is maintained. This node acts as a database of clients that are in the network. Using this network, the path and topography of the network is generated. The content is discovered by forwarding the file in the network. To the next attached node if the data is not available. Therefore, the node which has the content forwards a response to the central server. The following document explain the structure in which the code for the Unstructured peer to peer network is implemented with caching.

**Coding Logic:**

The code is implemsented in python using an event based approach. Meaning, as soon as a message is generated, the thread manager starts a new thread to handle the event as specified in the event list.

The flow chart explains the message flow:

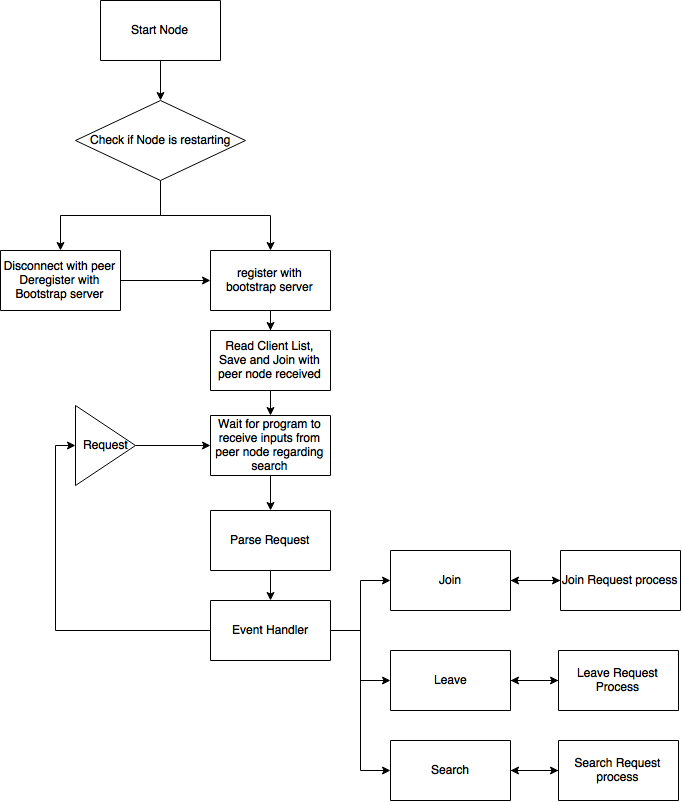


Figure Coding flow

As soon as the network is started, the bootstrap handler code connects to the bootstrap server and downloads a list of peer clients to connect to. The Node then attempts to use the client list provided to connect to the peer nodes in the network. Once the network connection is established, the clients then waits for the messages to be received for searching the files in the network. The event handler function identifies the message generated and then forwards the parsed message to the function to handle the message in a separate thread. In case the search message is received, the message is parsed and forwarded to the search handler

**Search Logic**

Search is implemented in the network with the help of a search ID. As soon as a message is received a search ID is assigned to it by the node receiving the query. Therefore, the messages propagating in the network will have a specific ID which will help identify the duplicates and also timestamp the messages received. In order to avoid flooding in the network, two approaches taken. The search response is sent by the node that has already received the message and does not have a client to forward it to that has not already received the message.

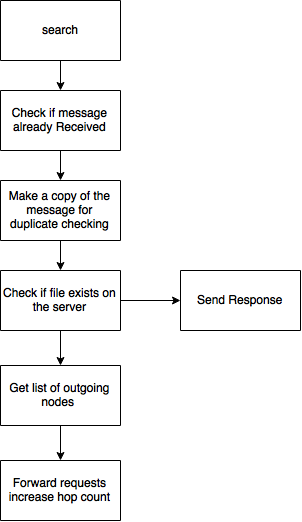


Figure Search Algorithm

**Caching Logic:**

Caching Algorithm implemented maintains the resource IDs of each of the files. As soon as a message is received, the resource ID count is incremented. The thread then checks if the message is the most popular query when the message response is received. If the message received has the highest rank in the node, then the address of the node is stored in the cache. This address is used during search to check if the message received is a popular query. If so, the message is then directly routed to the destination node which then sends a response back to the originator. The size of the cache is specified in the command line arguments.

**Code Implementation:**

The code is almost completely automated. The script is run as a daemon to start the process ds soon as the program is started, the client registers with the Bootstrap server and handles the registration errors. On Completion of the bootstrap registration, the client connects with the peer nodes and checks the responses. To enable easy maintenance and to audit the messages in the network, each of the registration, query generation, join requests and information about the node is maintained in a Redis database. Redis is a caching in memory database which allows fast data caching and provides support for publish and subscribe to channels. Each of the nodes have their own specific set of tables, therefore, it is still an unstructured network and does not interact with the other tables.

Start the client using:

python unstructpp [-h] -b BOOTSTRAP\_IP -p PORT -n BOOTSTRAP\_PORT [-u USERNAME] [-c cache\_size]

Code to run the query generator

python querygenerator

Auditing the nodes is done using a query generator that is run separately that will generate search queries to the network and collect the response. The script is implemented that will randomly select the nodes and forward random requests to the network by querying the bootstrap server based on zipfs distribution. Node degrees are calculated based on the results obtained by the program for the queries generated by the generator.