Programming Assignment 3

Design Document

Maintaining file consistency a Gnutella-styleP2P system

Arun Mathew Iype
Maheshwara Reddy A20284393

Contents

Introduction

This report details the design of the PA 3, Maintaining File Consistency in a Gnutella type file sharing system.

This project is built on the file sharing system built for the earlier programming assignment (PA 2), to build a de-centralized file sharing system. This project is programmed in C and has been built and tested on the Ubuntu OS.

All peers have access to a config file which has the list of all the peers to whom the system is connected. The peers are represented through their IP and Port (as [IP:PORT]). The file also specifies the type of consistency maintenance technique used, Push or Pull, and the "Pull Rate" for Pull technique.

All systems maintain two different directories to maintain the "Source" files and the "Downloaded" files. All the files that are created and modified in this system (Node), this is the Master Node for those files, and called "Source" files and are stored and maintained in the "Source" directory. The files that were downloaded from other Nodes are maintained in the "Downloaded" directory.

Any search from a Node (Client Node) is sent to all its peers and the peers in turn pass the query to their peers and the process continues until the TTL expires or all the Nodes in the system have received this message at least once. When a Node has the file requested in the query, it generates a hit-query and the hit-query is sent to the Client Node, in the same path taken by the query to reach this (Source Node) Node.

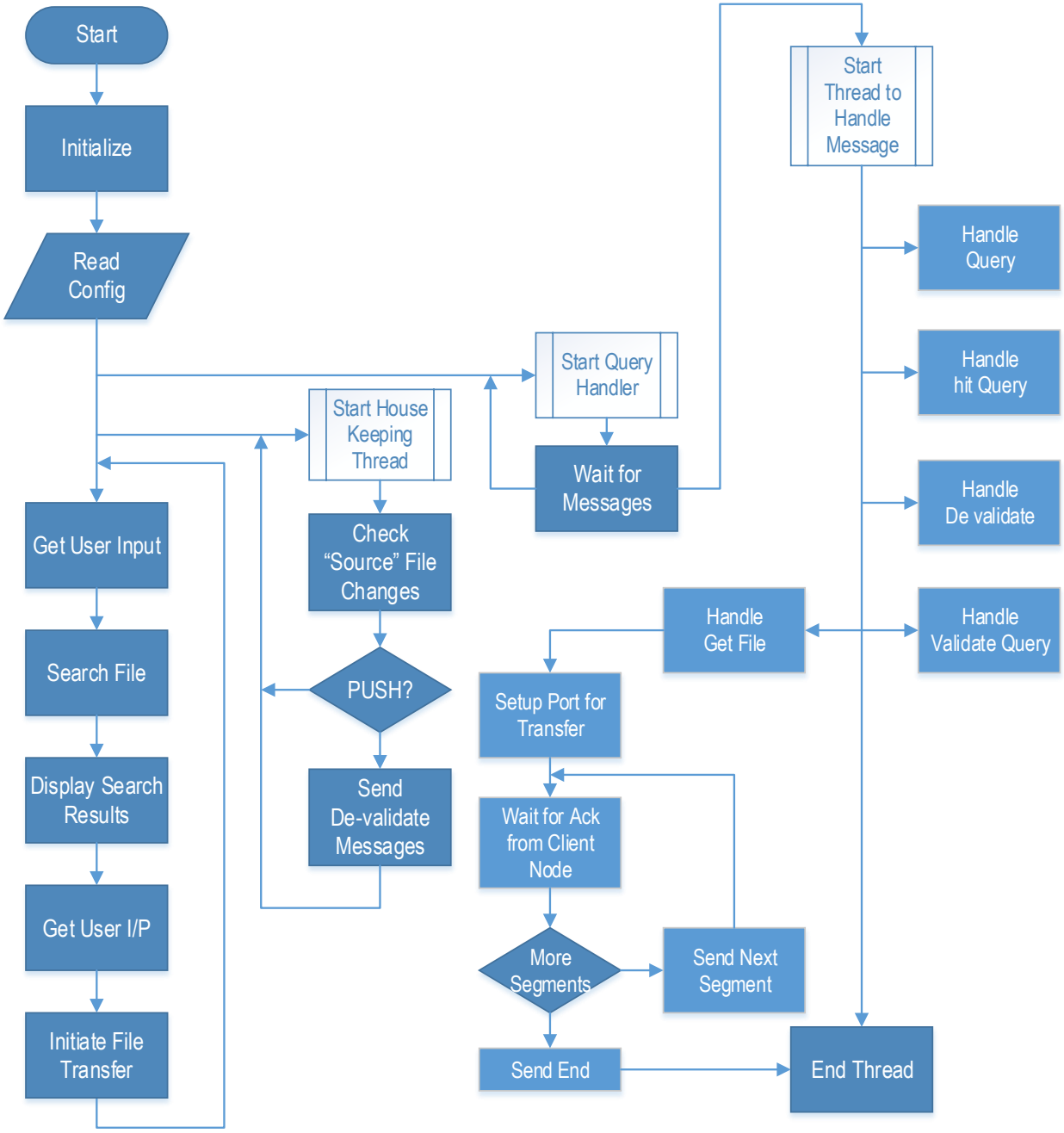
A Master Node of file will always reply with a hit-query for a query on that file, the other Nodes will reply with a hit query only if the file version is valid (is current).

When a Node receives a hit-query, for a query it had sent earlier, the Node records the reply. After a wait period of 10 seconds, the Nodes displays all the hit-queries received. The user can select the Node to the get file from.

On receiving a valid selection form the user, the Node sends a "GET" (Obtain) message to the Source Node. The Source Node allocates a dedicated port for file transfer and send a reply with the new port number and the count of number of segments in the file that is being transferred. Each segment is a block of 500 bytes (or less) of file data. A partial flow control is implemented through acknowledgements from the Client Node to the Source Node for every file segment received by the Client Node.

Design

Flow Chart



The above flow chart is a representation of the internal structure of the Node Program.

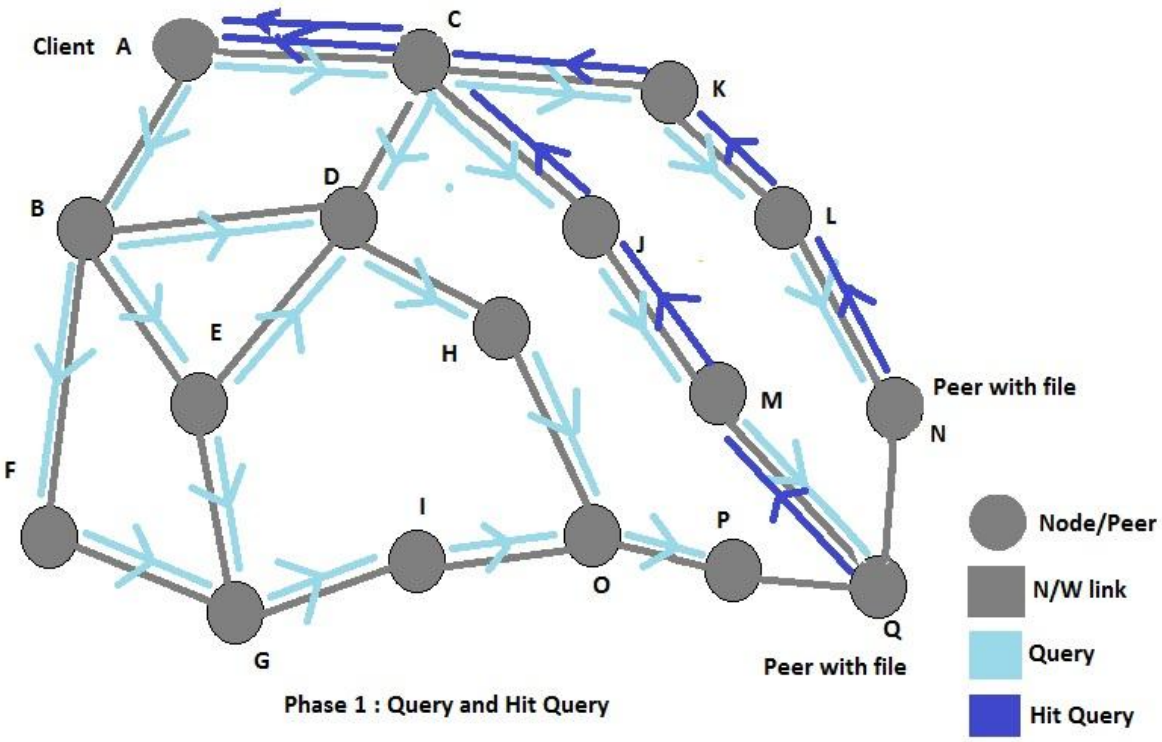
Data structures Used

1) seDe : Used to format messages for each session that the peer has seen.
    a. Messages stored with their message ids.
2) wDEx : Used to pass a structure to thread
    a. This helps in passing a uniform structure to the threads enabling modularity
    b. Threads can query the structure and extract the information they need
3) QhMap : Used to store the queryHit messages
    a. This stores where each message has come from
    b. This is used to send the hitQueries back to the source
4) QrMap : Used to store the query messages
    a. This stores where each message has come from
    b. This is used to send the hitQueries back to the source
5) Thread Pool : We maintain a set of pool of ready threads in the nodes, which can be quickly used

Message Formats

Message	Direction	Comment	Message Formats
Query	Peer to Network	Peer queries the n/w	**QRY[ttl]fileName[IP:Port:SlNo]IP:Port]**
Hit Query	Peer to Network	Peer which has the file replies	**HQY[IP:Port:SlNo]IP:Port][Origin_IP:Origin_Port]Version]**
Get File	Client Peer to Server Peer	Client Peer contacts Server Peer for the file	**GET[fileName]IP:Port]**
Get Reply	Server to Client	Server Peer responds with details of the file	**GRPY[No of Segments][Port]**
Obtain File	Client to Server	Client peer requests for the file segment by segment	**OBTAIN**
File Transfer	Server to Client	Server sends the segment requested	**FTRAN [segment data**
End Transfer	Client to Server	Server sends end of file transfer message	**ETRAN**
De validate Message	File Master to all	Devaluate messages from Source when the files are modified	**DVL[IP:Port:SlNo][Origin_IP:Origin_Port][fileName][Version]**
Validation Requests	To File Master	Message seeking for validation of a file version	**VLQ[IP:Port:SlNo][IP:Port][fileName][Version]**

Peer Design



Phase 1 : Query and Hit Query

The peer can behave both as a client and as a server while receiving or sending files. The peer listens to a port and on receiving a message creates a new thread for processing it. The new thread interprets the message. It then does the necessary processing like sending a reply message or append the received file bytes to the file.

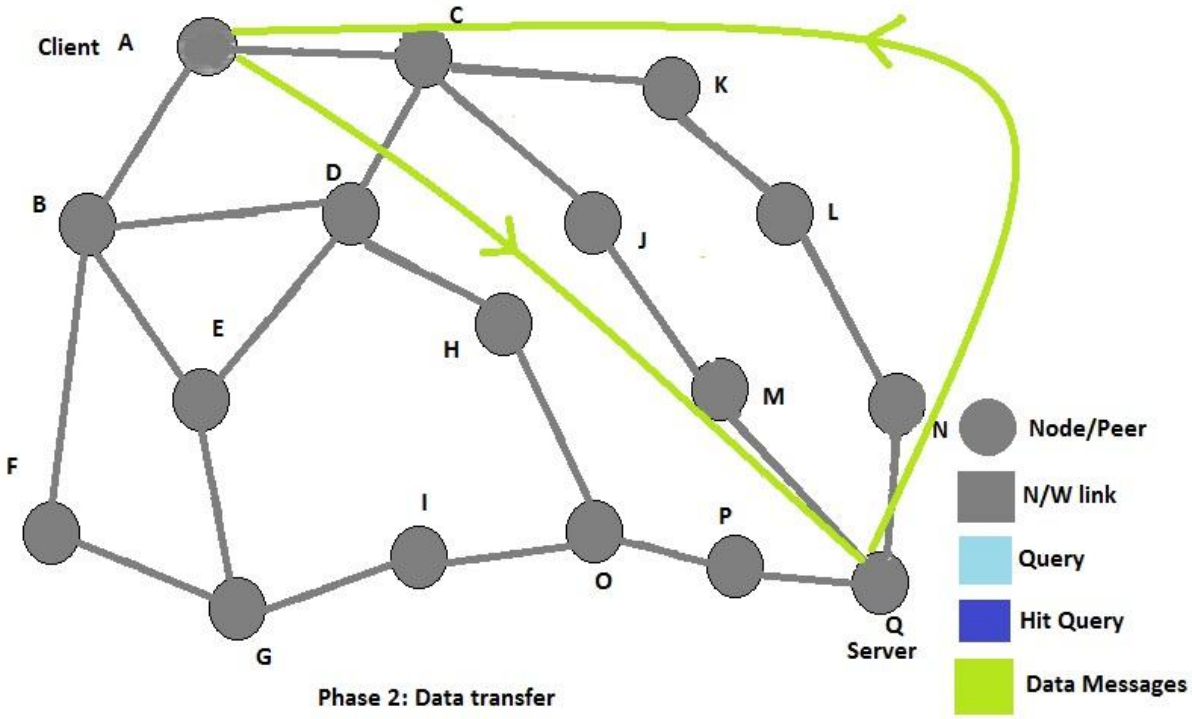
Phase 1 of the getting a file involves finding who in the network actually has the file. In the figure shown above the network in a sort of overlay structure in which each peer knows it neighbors. In effect all the peers are in the same machine, the lan or the internet. Communication protocol used is TCP-IP.

In the example shown above, node A wants to search for a file. It sends a QUERY message (shown by blue arrows) to its neighbors B and C. B and C in turn forward this message to their individual neighbours other than A. In this way the original message floods the overlay network. Each message has a TTL value set by the initial sender. Each node when it forwards the message decrements this TTL value. This helps prevent the message from going around the network endlessly. In the above figure, node P cannot forward the message to node Q as the TTL has expired.

Each node also maintains a list of messages that it has forwarded. This also helps in preventing messages travelling in circles. When a node finds that it has the requested file, it sends a HIT QUERY message (shown by green arrows) to the original first sender through the same route.

Since each node has a list of messages it has received and disposed, this back route is easy to get. In the above figure, both node N and Q have the file with the same name with them. So they both send the HIT QUERY reply back up to A.

At the end of phase one, the client is aware of the ip and port of possible servers ie N and Q in our scenario.



Phase 2: Data transfer

In Phase 2, the data transmission actually begins. The client must select one peer to act as the server. After this client peer and server peer communicate directly with one other ignoring the overlay structure. The data pattern is shown in the figure below.

Get Message : Client requests the server to send the file details

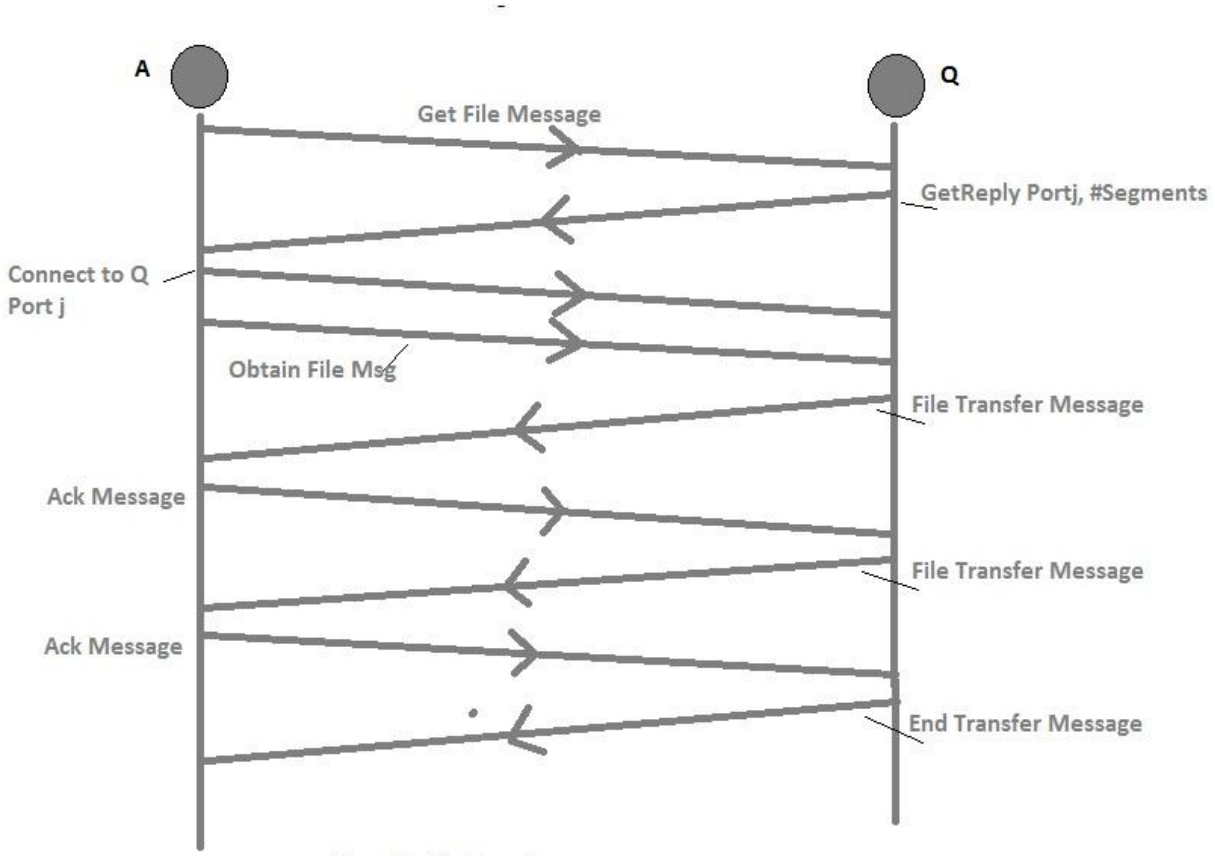
Get Reply Message :Server sends the file details such as number of segments the secondary port to be use for file transfers

Obtain file Message : Client sends an obtain file message with the segment number which it needs.

File Transfer Message : Server sends the file as a sequence of bytes to the client

Ack Message : Client indicates that it has received the segment and to send the next segment number

End Transfer Message : Server indicates that this is the last file segment



Phase 2: File Transfer

File Consistency - PUSH

The file Consistency is maintained in the PUSH by the file Master periodically monitoring changes in its "Source" files and sending de-validation messages to all the Nodes in the network.

On receiving a De Validate message for file, all Node with the file in their "Downloaded" directory will mark the file as invalid.

When Nodes processing a Query request from a peer, check if they have a "valid" version of the file in their "Downloaded" directory. A hit-Query message is sent only if the file is valid.

File Consistency - PULL

The file Consistency is maintained by the Nodes that have Downloaded a file from the network. Periodically these Nodes send a "Validate" query for each of these files to their Master Nodes.

On receiving a "Validate" query from a Node, the Master Node sends a de-validate message to the source of the message if the file version is incorrect.