**CS 550**

**Programming Assignment 2**

**A Simple Gnutella-style P2P File Sharing System**

**Fall 2013**

By

Maheshwara Reddy: A20284393

Arun Mathew Iype : A20278285

Contents

[Introduction 2](#_Toc368920946)

[Peer Design 2](#_Toc368920947)

[Data structures Used 10](#_Toc368920948)

[System design 10](#_Toc368920949)

[Basic communication design 10](#_Toc368920950)

[Messages 10](#_Toc368920951)

[Message Format 11](#_Toc368920952)

# Introduction

In this project we have tried to implement a p2p file sharing system. We have used C language for programming and Berkeley Sockets for the communication between the peers.

All peers have access to a config file which lets them know the ip/port of all the other peers. The idea is those peers share the files in their working directory.

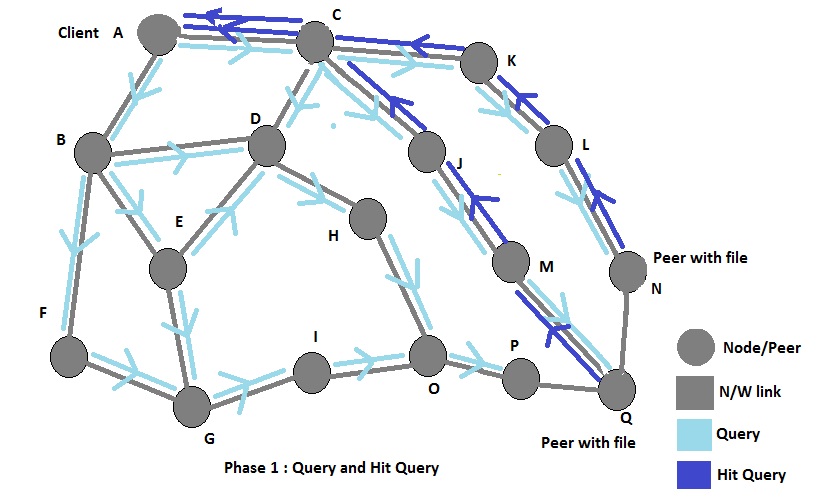
Any peer when it searches for files behaves like a client. When a client wants a file, it sends a query message to all its neighbors. The neighbors in turn forward this message to their own neighbours. When the peers, which has this file receives the search message, it sends back hit query message.

The hit query message travels back the same path to the original client. The hitquery includes the ip port of the peer which has the file, which will now act as the server. The client used the info in the hit query to query the server and receive the file in chunks.

Design

## Peer Design

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.

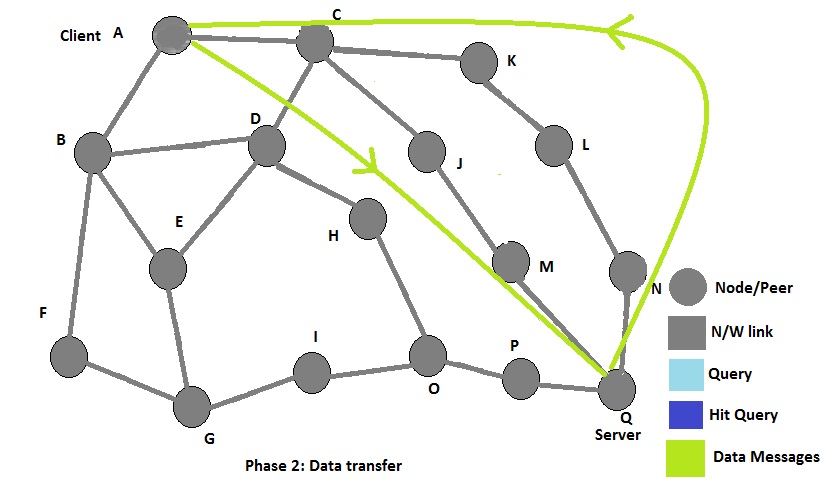


Phase 1 of the getting a file involves finding who in the network actually has the file. In the figure shown above the network in a sort of overlay structure in which each peer knows it neighbors. In effect all the peers are in the same machine, the lan or the internet. Communication protocol used is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

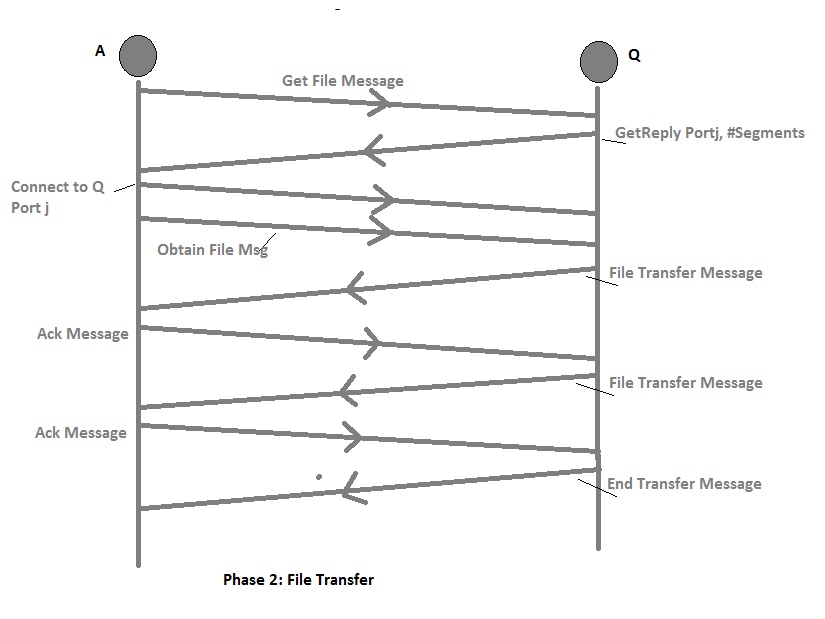
In the example shown above, node A wants to search for a file. It sends a QUERY message (shown by blue arrows) to its neighbors B and C. B and C in turn forward this message to their individual neighbours other than A. In this way the original message floods the overlay network. Each message has a TTL 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.



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.



Phase 2 involves only two peers instead of the entire network. Through a sequence of requests and responses the file is downloaded chunk by chunk. The communication here is done through a secondary port on both the server and client.

**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

Start

Start Thread to Handle Requests Port 0

Start Daemon Thread for Housekeeping

Housekeeping Thread

Get User Request

Send query message to neighours

Wait for 1min for replies

Read Config file

Initialization

Handle Request Thread

Process Request Thread

Create Thread

Wait for Request

Handle Request

Send Get Message

Process Request Thread

If Query

Query Process function

If Hit Query

Hit Query Process function

If Get

Get File Thread

Exit Thread

Send Hit Query

File Found

Check file list for file

Forward Query message to neighbors

Query Process function

Forward hit query to original message sender

Message ID Present

Look up Message ID

Stop

Hit Query Process function

Get File Process function

Find File Stats

Create New Port

Send message “Get Reply” to Client

While all Segments not sent

Send ETRAN

Exit Thread

Wait for Obtain

Send Segment

## Data structures Used

1. seDe : Used to format messages for each session that the peer has seen.
   1. Messages stored with their message ids.
2. wDEx : Used to pass a structure to thread
   1. This helps in passing a uniform structure to the threads enabling modularity
   2. Threads can query the structure and extract the information they need
3. QhMap : Used to store the queryHit messages
   1. This stores where each message has come from
   2. This is used to send the hitQueries back to the source
4. QrMap : Used to store the query messages
   1. This stores where each message has come from
   2. 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

## System design

We have done the implementation and testing on Ubuntu Linux using gcc compiler.

## Basic communication design

* Communication is implemented using Berkeley Sockets. The peers listen to ports continuously. On receiving a connection the create a new thread or process to handle the request and go back to listening on the port.
* Peers : have two listening ports. Port0 is used for normal listening and Port1 for file transfer related listening and sending. Port 1 is activated only when a file transfer starts and disabled when its over.

Peer 0

Port 0

Port 1

Peer 1

Port 0

Port 1

Fig 4. Listening ports for the peers

### 

### Messages

|  |  |  |
| --- | --- | --- |
| Message | Direction | Comment |
| Query | Peer to Network | Peer queries the n/w |
| Hit Query | Peer to Network | Peer which has the file replies |
| Get File | Client Peer to Server Peer | Client Peer contacts Server Peer for the file |
| Get Reply | Server to Client | Server Peer responds with details of the file |
| Obtain File | Client to Server | Client peer requests for the file segment by segment |
| File Transfer | Server to Client | Server sends the segment requested |
| End Transfer | Client to Server | Server sends end of file transfer message |

### Message Format

|  |  |
| --- | --- |
| Message | Format |
| Query | QRY[ttl]fileName[IP:Port:SlNo]IP:Port] |
| Hit Query | HQY[IP:Port:SlNo]IP:Port] |
| Get File | GET[fileName]IP:Port] |
| Get Reply | GRPY [No of Segments][Port] |
| Obtain File | OBTAIN |
| File Transfer | FTRAN [segment data] |
| End Transfer | ETRAN |