Network-Centric Computing - CS 382

Assignment 1: Napster

Hassan Shahid Khan, Wajih Ul Hassan, Gohar Irfan Chaudhry

Due Date: 16th Feb. 2015 11:55 pm

Submission Instructions:

- Please use the following naming convention: <Roll# Person 1>_<Roll# Person 2>_Assignment1.zip
- **We will NOT accept any submissions via e-mail.** There will an LMS tab for late-submissions as well. There are no exceptions to this rule.
- Be prepared to justify the code you have submitted in vivas. We take plagiarism concerns very seriously in this course.
- You need to work on this assignment in groups of 2.
- Please direct all questions related to the assignment to the discussion forum at Piazza.
- Late submissions will be penalized 10% per day.

Introduction:

Hello and welcome to the course! This assignment is going to introduce you to the fundamentals of socket programming. We will be developing a simple peer-to-peer system which will be similar to Napster in terms of functionality. You can find further information on how Napster works on the following website: http://computer.howstuffworks.com/napster2.htm .

Programming Languages:

Although we will try to keep implementations open-ended in this course, we strongly suggest you use one of the following languages for developing your application:

- Java
- Python
- C/C++

No GUI's are required for this assignment simple CLIs (Command line interfaces) are sufficient.

Application Details:

Your primary task in this assignment is to create a P2P (peer-to-peer) file-sharing network application similar to Napster. This application has two core components:

1) A Central Indexing Server:
   This server will keep track of which client stores what file(s). It essentially provides search facility to clients. In our case simple file search using their file names.

2) A Client/Peer:
   Just like Napster, the application must be able to handle multiple clients. Each client can have an arbitrary number of files stored with it.

Client Interface:

Your client application must be able to handle the following commands:-

- *Lookup <File-Name>*: This command (directed towards the central server by default) should return the IP-address(es) of the clients at which the target file-name resides. In case the file is not found on the server, it should return with either NULL or -1.
- *Get <File-Name> <IP Address>*: This command should fetch a copy of the target file from the specified IP address and add it to the clients' local repository. As above, return with NULL or -1 in case the file is not found at the IP address.
- *Put <File-Name> <Data To Be Written>*: This command should create a file in the clients' local repository and forward this update (i.e the name of the newly created file) to the central server. Upon successful receipt of this update by the server, it should return +1 to indicate the success.

Protocol:

- Whenever a new client is created, it must register itself with the central server.
- When a client either connects to the central server or when the put command is issued, it should inform the server of the file names, but should not actually transmit file-data to the server.
- If a client does not have a file (queried by the lookup command) in its repository, it should send a request for that file to the central server. The server has a list of other clients IPs with along their file listings. The response from the server should allow the client to fetch the desired file directly from the client that has the target file.

Note: You are free to use either TCP or UDP for network transmission in this assignment.

Hints:

- Start off by writing a simple server-client application which can exchange messages. Only start implementing actual transmission of file-data once you have this structure in place.
- If you are confused as to how multiple client applications can run at the same time, you can do this by binding each client to a different port. It would be very convenient for your client application to take in <port_number> as an initial argument when it is created.

Bonus:

Parallel download feature

Use multi-threading to allow multiple clients to download different files from a single target client at a given time *in parallel*.