**CPS 706 System Architecture**

By: Rudi Zhou – 500681845

Eric Jaskierski – 500644178

Ajay Herod - 500764104

Team 2

Presenting with Haytham Qushtom

System.doc containing the description of the architecture of the application which describes data structures, design choices, and important algorithms used.

For the code written to implement the Directory Server, we used the resources given to us to create this implementation using a DHT or Distributed Hash Table. For our hash table we created it using Strings for the key and value pairs to represent the ID’s 1-4 and connect them in a ring as specified in the project notes. In the Directory Servers, we also implemented a multi-threaded application by using multiple threads. In each case there was a main thread for both the TCP and UDP socket connections and then whenever a upload/download request is made, the method findAvailable(UDP/TCP)Port is called to create a new thread and find the next available or open port number starting from port 0.

For the DirectoryServer,java and P2Pserver.java, we also decided to add final variables to represent the HTTP code and the FINAL choice was chosen since these codes will never change. When error checking any of the upload or download portions to see if they fail (wrong filename, filename doesn’t exist, etc) an appropriate error or HTTP error code will get spit out which simulates the experience of a real HTTP1.1 protocol.

Besides that, most of the data containing the files were put into byte streams and all the byte data was stored in arrays. Nearly everything else such as the filenames, server IP addresses and port numbers, were also stored in arrays since the scale of the project is not that large. We do not expect there to be a lot of Directory Servers (4) and the number of images files used for the demo is around 2-3. This means we can know how much data is needed to store information so a dynamically allocated data structure such as a LinkedList is not really needed. Order also doesn’t matter too much so the other advantage of LinkedList’s (ease of deletion / insertion) is also not needed so Arrays were the most suitable data structure for this project to store information.

For the Download functions, we decided to break the files down into a byte stream and read them from a Directory Server socket. We then initialize a FileOutput and Buffered Stream reader and if the client can successfully connect to the Peer Server, it will read the file from the server and then create a byte array to receive the file by writing down bytes read into the client machine.

The Upload function works similar to the Download function except in reverse order where a client will upload a file of their choice and then if it is successful it will send the message over to the server using method sendDataToServer and then create a separate message based on the response from the server from method receiveDataFromServer. This method will return an HTTP code based on the initial message sent and if a 200 OK message is received then we know that the file was successfully uploaded into the server and thus the application will spit out a message stating file has been received by server and upload was a success.

The query function was made to simulate an HTTP GET request so we will simply generate a String that mimics a GET request. The initial message will be sent with a string to match if the file exists in the Server or not. If the requested query string is not found in the server then we know client probably mis-typed the file they were looking for and thus we will return the appropriate HTTP error code. If the file is successfully matched with the query string request, then we know file is found and we will construct the actual HTTP GET request and send it over to the server. If a 200 OK response is sent back, then we know the request successfully got through to server and thus we will return message stating that the file was found.