# CSE 514 LAB 1

# Descriptions

## Protocol Overview

The protocol for communication between the clients and server revolves around a single serializable object. The serializable object contain attributes that allows the receiving host to determine the type of commands that the sender wants. To reply to the receiving message, the receiver would create a local serializable object, place the data and command to send back in the proper fields of the object, and send it back to the client.

The same serializable object is used for client to server and client to server communication. By using a single object as the primary mean of communication between the client and server, it allows the program to be easily extended in the future. To add features to the program, the future developer would just simply need to add additional fields to the program.

## Client to Server Communication

### Overview

The relationship between the client and server is that the server act as a tracker. It keep track of all the peers that has the file, all new peers to join, and remove the peer once they leave. Other than meta-data about the available files and clients, the server does not host any other data. There are several reasons why the client would contact the server and they are:

* To register a file
* To get a list of available files to download
* To request the locations of the file, meaning to get a list of all the seeds for the file
* To request to leave

### Communication

Communication between the peers and the server is very simple.

A single serializable class contains all the commands and interfaces needed between the server and the client. Whenever this object is being serialized and passed onto the receiving end, a command is given and also additional information that will help the receiver perform the action dictates in the command.



Below are the tables of the commands and the data that they carry with them. For instance, when the receiver deserialized the message object and see that the command that the client had sent was a Register Request, the server knows that it can obtain the information that are relevant with this command by looking at the ClientIP, ClientPort, NoOfFiles, Files, and FilesLength.

**Commands from client to server**

|  |  |
| --- | --- |
| Register Request | |
| Client IP |  |
| Client port |  |
| Number of Files |  |
| Name of the files | This is a list containing the name of the files that the client wishes to register |
| Length of each of the files | This is a list containing the length of all the files |
|  |  |
|  |  |
| File List Request | |
| Client IP |  |
| Client Port |  |
|  |  |
|  |  |
| File Location Request | |
| Name of the file | This is the name of the file that the client wishes to obtain |
|  |  |
|  |  |
| Leave Request | |
| Client IP |  |
| Client Port |  |
|  |  |

**Commands from Server to Client**

|  |  |
| --- | --- |
| Register Reply | |
| List of booleans | This tells the client which of the files were successfully registered |
|  |  |
|  |  |
| File List Reply | |
| List of the files | The server will exclude any files that the client already has |
| Number of files |  |
| List of file lengths |  |
|  |  |
|  |  |
| File Location Reply | |
| Size of the file |  |
| No of end points |  |
| List of IP addresses | These are the IP addresses of the clients that has the file |
| List of Ports | These are the port numbers of the clients that has the file |
|  |  |
|  |  |
| Leave Reply | |
| None |  |
|  |  |

## Client to Client Communication

Communication between client and client follows the exact same means as information between server and client. The two additional commands in the Server Client Message object, Data Request and Data Reply, provide clients to request files segments from each other.

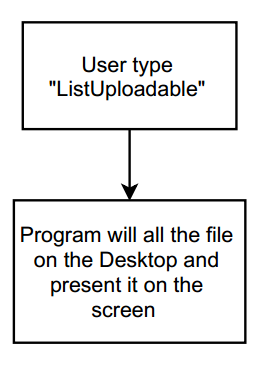
|  |  |
| --- | --- |
|  |  |
| Data Request | |
| Name of file | This is the name of the file that the client wishes to obtain |
| Segment of file | This indicates which segment of the file the server should send |

|  |  |
| --- | --- |
|  |  |
| Data Reply | |
| Hash for the file | This is sh 256 hash |
| segment of file | This tells the client which segment this is |
| Resulting data segment | This is data segment to send back |

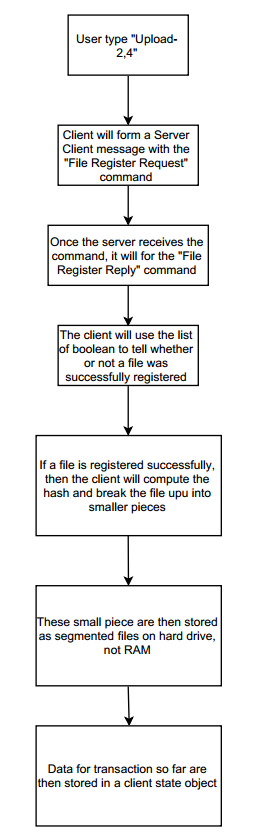
## Controls

The program is a console application and uses the user’s typed input to perform the corresponding action. The following are the flow chart depicting the commands that the user can enter and the response that he/she would get.

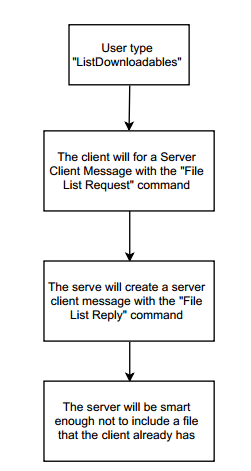
**“ListUploadables” Command**



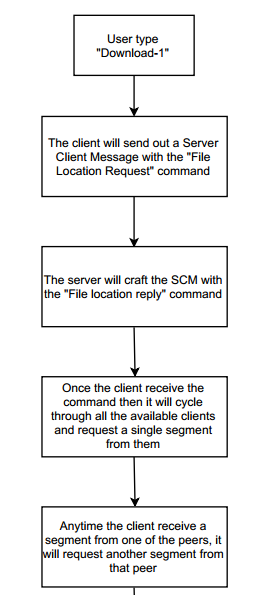
**“Upload-##” Command**

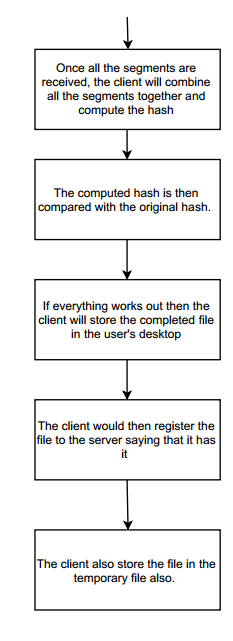


**“ListDownloadables” command**

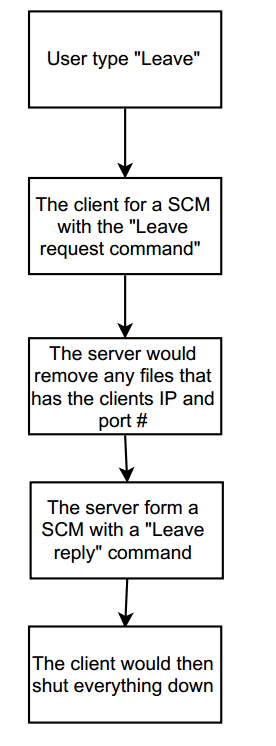


**“Download-#” Command**





**“Leave” Command**



# What works and doesn’t

## Working

* Interfaces: All the interfaces required are implemented and are fully functional.
* Parallelism: Communication between the server and clients are 100% parallel. The server will support up to 40 parallel connection at a time.
* Downloading from multiple peers – The program can download file segments from multiple peers and recompiling all the segments back together once all the segments are obtained.
* Error checking – Sh-256 Hash is performed before the file is registered to the server and is checked again once all the segments are received by a host.
* Download completion – When a file is done download, the client will automatically register the client as a potential seed for that file.
* Storage of the file – The resulting files are successfully stored onto the desktop.
* Successful removal of the peer when the peer leaves the network

## Doesn’t work

* Failure tolerance mechanism – No failure tolerance mechanism were implemented
  + Potential solution – although I was not able to implement a failure tolerance mechanism due to time constraint, a potential solution for it would be to require all the clients to register to the server at a regular interval. If a client does not check in with the server within a certain time frame, then the server can reasonably assume the client is disconnected and it can remove that particular client its tracking system.
* Status – The program does not support the display of the percentage of the file the host has received so far.

# Structure of source code

The structure of the source code can be seen by understanding the responsibility of the various classes.

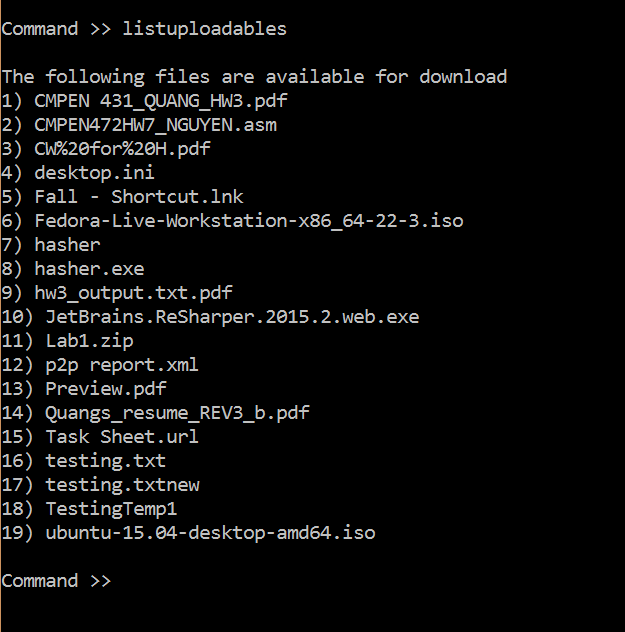
* Program
  + This class obviously contains the function for the program. The responsibilities for this class are:
    - If this is a server program
      * Instantiate the server.
    - If this is a client program
      * Instantiate the client’s server
      * Reading the user’s input
      * Instantiate a TCP client (TCPTorrent class) to carry out the user’s command
      * Depending on the user’s command, the Program class may display additional information to the console screen
* TCPTorrent
  + This class contains the methods that allow the main program to instantiate both the client and/or server. This is class is responsible for carrying out all the network communication.
* ServerClientMsg
  + This is the serializable object that is being passed around between the client and server, or client and client.
* ServerDataObject
  + This class allows the server to maintain all the data needed for the server to act as a tracker. Information such as client’s IP and Port, and files are stored in this object.
* StateObject
  + Similar to how the ServerDataObject maintain the state of the server, the StateObject maintain the state for the client.
* ClientPassableObject
  + This class allows the main program to instantiate a TCP client that is able to carry out the proper command.
* DataSegmentObject
  + This class is responsible for splitting up the files, producing the hash code, and putting the files back together.

# Sample output

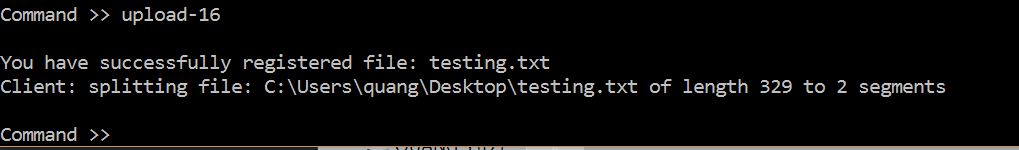
## Client #1

In this case, the client fires the program and decided to upload a file. See below for the command to bring up the list of files to upload and the command to upload file #16.

**Command to list all the files in the Desktop folder**



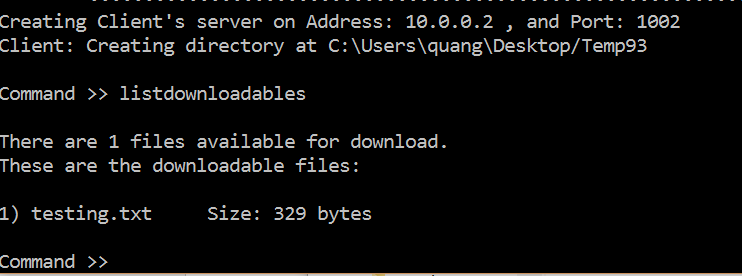
**Command to register file #16 to the server**



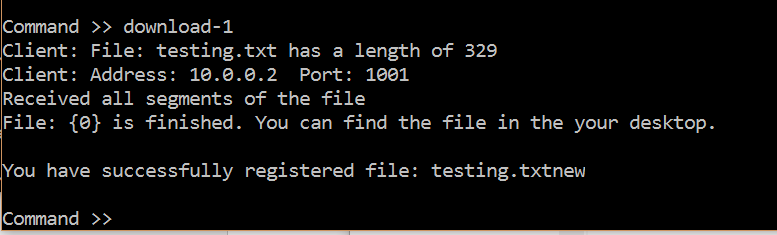
## Client #2

In this case, client #2 just enters the program and he/she query to see if any files are available for download. Since he/she sees that the file Testing.txt is available for download, he/she will download it. Below are the commands to do so and the output associate with it.

**Client #2 request for the list of downloadable files**



**Client #2 downloading “testing.txt”**



## Leaving

Lastly, when a client wants to quit, he/she can use the “leave” command. Please see below for an example of the leave command

