Project Report For Implementing Peer to Peer File Down-loader

Name: ATUL GARG
UFID: 94432505
Email: atgarg@cise.ufl.edu

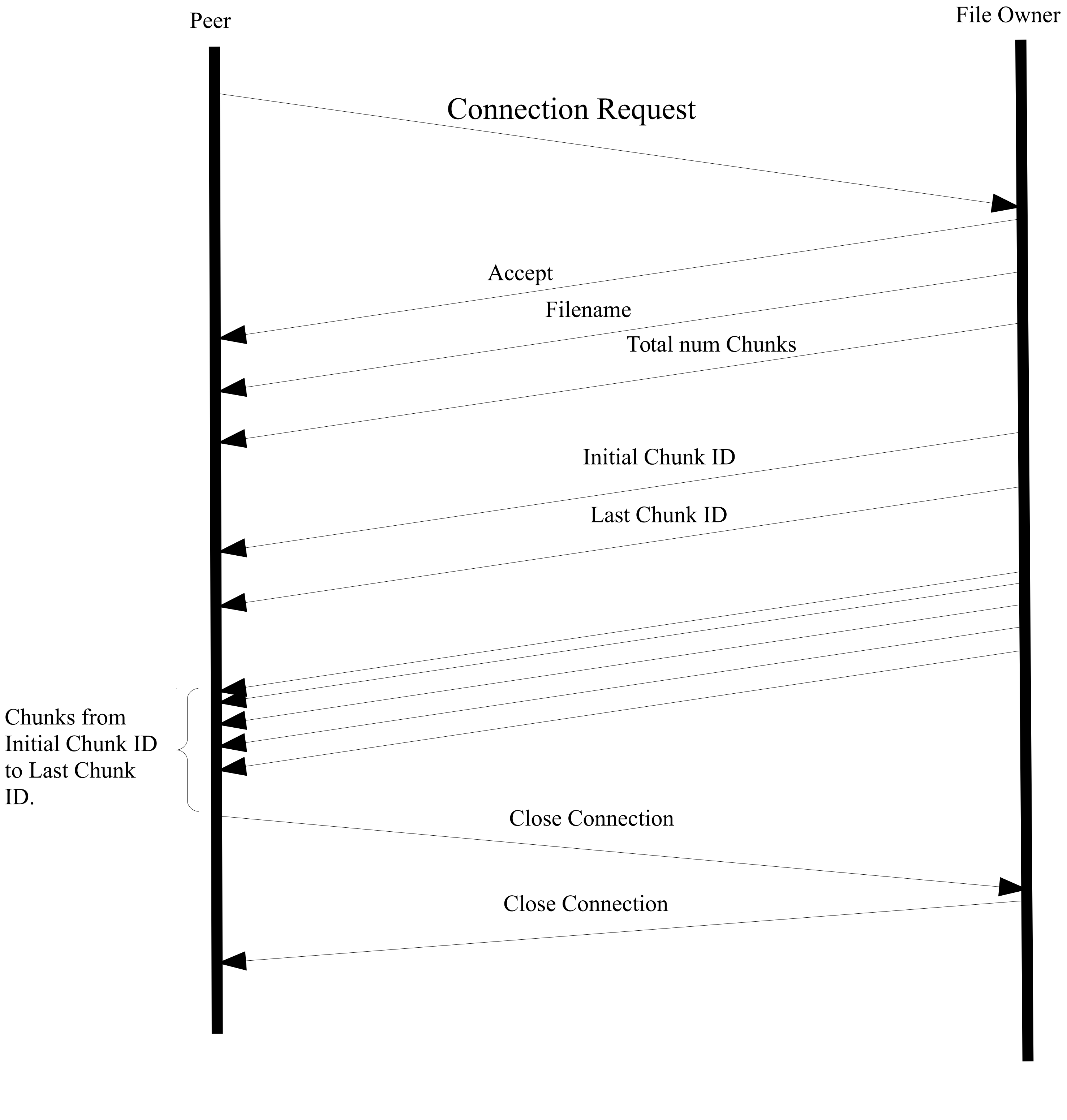
Problem Statement

This project creates a peer-to-peer network for file downloading. It resembles some features of Bit-torrent, but much simplified. There are two pieces of software – peer and file owner. The file owner has a file, and it breaks the file into chunks of 100KB, each stored as a separate file. The minimum size of the file is 5 chunks. The file owner listens on a TCP port. It should be designed as a server that can run multiple threads to serve multiple clients simultaneously.

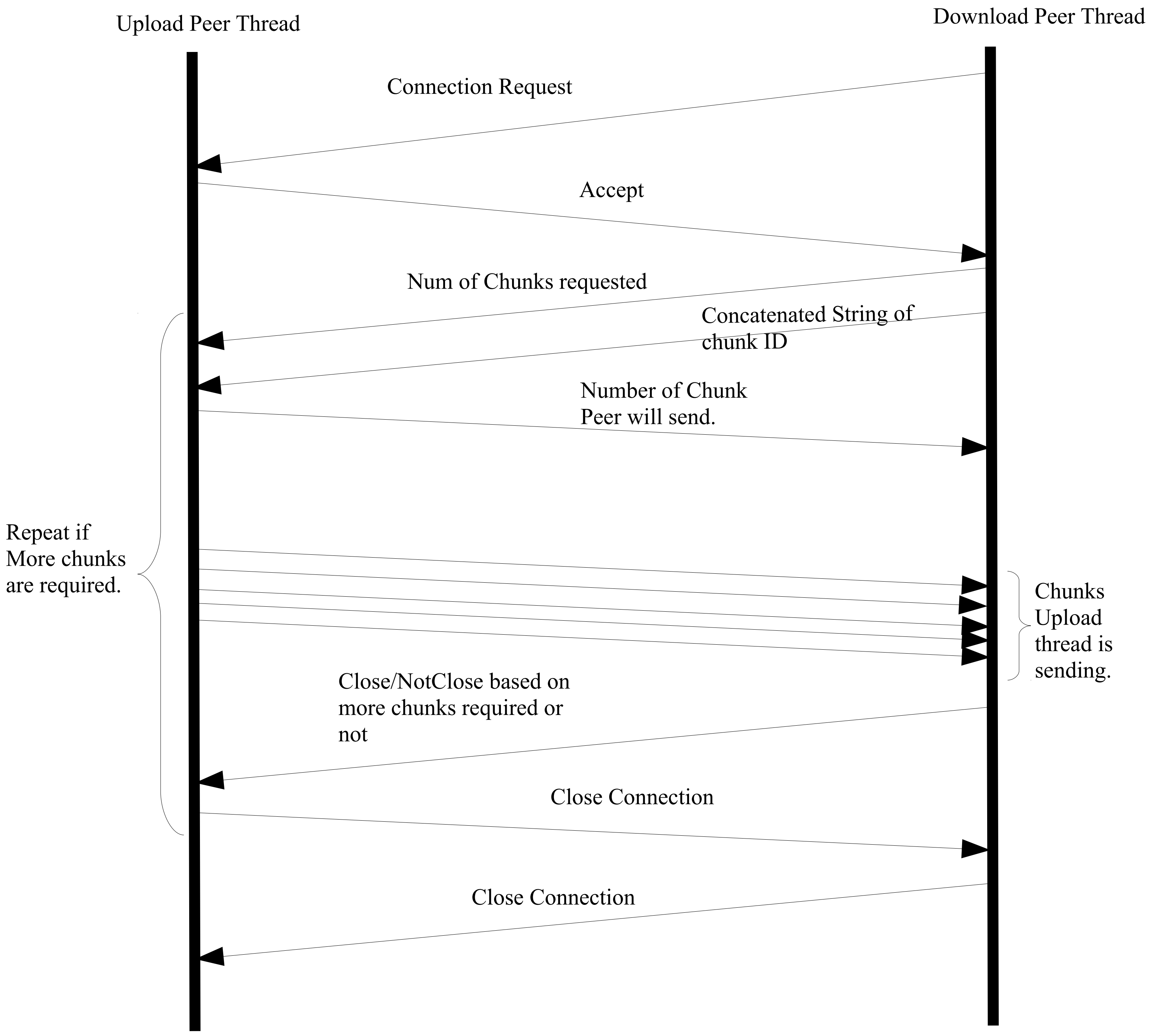
Each peer should be able to connect to the file owner to download some chunks. It then should have two threads of control, one acting as a server that uploads the local chunks to another peer (referred to as upload neighbor), and the other acting as a client that downloads chunks from a third peer (referred to as download neighbor). So each peer has two neighbors, one of which will get chunks from this peer and the other will send chunks to this peer. You can arbitrarily decide on the neighboring relationship as long as the network is connected --- there is a direct path from any peer to any other peer. The neighboring relationship may be encoded through input parameters (see below).

1. Start the file owner process, giving a listening port
2. Start five peer processes, one at a time, giving the file owner's listening port, the peer's listening port, and its download neighbor's listening port.
3. Each peer connects to the server's listening port. The latter creates a new thread to download one or several file chunks to the peer, while its main thread goes back to listening for new peers.
4. After receiving chunk(s) from the file owner, the peer stores them as separate file(s) and creates a summary file, listing the IDs of the chunks it has.
5. The peer then proceeds with two new threads, with one thread listening to its upload neighbor to which it will upload file chunks, and the other thread connecting to its download neighbor.
6. The peer requests for the chunk ID list from the download neighbor, compare with its own to find the missing ones, and download those from the neighbor. At the mean time, it sends its own chunk ID list to its upload neighbor, and upon request uploads chunks to the neighbor.
7. After a peer has all file chunks, it combines them for a single file.
8. A peer should output its activity to its console whenever it receives a chunk, sends a chunk, receives a chunk ID list, sends out a chunk ID list, requests for chunks, or receives such a request.

Protocol Used Between File-Owner and Peer.



Protocol Between Peer to Peer Communication


Upload Peer Thread
Download Peer Thread
Connection Request
Accept
Num of Chunks requested
Concatenated String of chunk ID
Number of Chunk Peer will send.
Repeat if More chunks are required.
Chunks Upload thread is sending.
Close/NotClose based on more chunks required or not
Close Connection
Close Connection

Implementation

<u>**Classes For File Owner.**</u>

Class FileOwner
Class FileOwner for Server to upload file to peers. Class implements Runnable so to execute each client request as a separate thread.
Methods:

- **public void splitFile(File file,String fileName)**
  method splitFile to divide the file in chunks. File is minimum divided in 5 parts. Each part is minimum of 100KB.
- **public void deleteChunks(String fileName, int numChunks)**
  deleteChunks to delete all the temporary files created for sending across the network.
  @param String fileName name of the file.
  @param int numChunks Number of chunks of the file.
- **Public void main(String[] args)**
  All driver code is written in main. It creates a server socket and starts listening at specified port. It creates threads for each connection of Class Connection where transfer of chunks from server to Peer happens.

Class Connection
(Implements Runnable)
Class Connection Defined for each connection established with client. Initialized using port specified as accepting port and is used to send and receive data to client. Implements Runnable to implement threads and method run is over ridden.
Methods:

- **public void run()**
  Method run() overridden to run as a thread for each client. Method decides a protocol between peer and server and based on the protocol sends the filename, total number of chunks, initial and last chunk id for chunk sequence and chunks itself in order.

<u>Classes For Peer.</u>

Class PeerBase
Class PeerBase comprising basic utility functions of Peer Object functions. Class used for extending.
Methods:

- **public void sendChunk(String fileName,ObjectOutputStream out)**
  method sendChunk to send a chunk specified by name as filename over output stream out. Method uses a predefined buffer size for communication over the network.
- **public void saveChunk(String chunkName,ObjectInputStream in)**
  method saveChunk to save chunk from a Input Stream in and save it in a temporary file by name specified by chunkName.
- **public void deleteChunks(String filename,int numChunks)**
  method deleteChunks to delete all chunk files after the merge is done.Method sticks to nomenclature pattern of naming all the chunks with format of "filename.chunk.numChunk".

Class Peer
Driver Class for Peer. Provides functionality to connect to FileOwner by specifying IP and Port number. Later to create threads which will act as Download thread to download from Peer and Upload thread to upload to peer.
Methods:

- **public void start(int serverPort,int peerListeningPort,int downloadPeerPort)**
  method start to establish connection with server and recieve chunks from server and store it as a files. Later start with two new threads one responsible for accumulating other chunks for download neighbour and other for uploading chunks that peer has but upload neighbour does not have.
  @param serverIP String format Server IP
  @param serverPort Port number for FileOwner.
  @param peerListeningPort port at which this peer will listen.
  @param downloadPeerPort port at which this peer will connect to download.

Class PeerUtil
Class PeerUtil to pass reference to data object to threads. Class Object is shared among DownloadPeer Thread and Upload Peer Thread.
Members and Methods:

- *Members:*
  HashMap<String,Boolean> chunksMap; //Used for Table look up for chunks present or not
  String fileName; //Name of the file.
  int totalChunks; //Total chunks of the file by FileOwner.
  int downloadPeerPort; //Port this Peer will connect to for download.
  int uploadPeerPort; //Port this Peer will run as server.
  int chunkSize; //Predefined chunk size.
  int chunksReqd; //Number of chunks required.
  int numChunkSend; //Number of Chunks Peer will send.
  int numChunkRecv; //Number of chunks Peer will recieve.

- **private void initialiseChunkMap()**
  Method initialiseChunkMap private method to initialize chunkMap to set all chunk value to false.
- **public Boolean check()**
  Method check to check if more chunks are required or the file has been completely downloaded.

Class UploadPeer
Class Upload Peer For Peer to act as Server in P2P network and upload chunks to requesting peers. Protocol so implemented is Upload Peer server runs on a predefined port which is passed as a part of Constructor. Next it accepts a connection from a Peer in form of FileName and chunkid. Upload peer checks if the Server already has chunks requested. If chunks are found then Server sends the chunks and waits for "CLOSE" message from Download Peer. If Chunks not found then Server puts the thread on sleep for the while it gets those chunks.
Methods:
- **public void run()**
  Method run implements all protocol for communication between peers.
- **private String[] splitMessage(String msg)**
  Method splitMessage to split the message recieved from neighbour. As per protocol it expects message to be of comprised of chunkname separated by “^”. Returns a array of all the chunk names required by the peer.
- **Private void sendChunks(String[] chunkList,ObjectOutputStream out)**
  Method sendChunks acts as wrapper around sendChunk and sends all the chunks which are requested by the peer and are present with this peer.

Class DownloadPeer
Class DownloadPeer responsible for downloading all the chunks of the file not yet downloaded. Download Peer will act as a client and connect to port specified. It will then try to fetch all chunks it can download from the Peer.
Methods:
- **public void run()**
  Method run implements all protocol for communication between peers. Method run implemented since Download peer will be invoked as a thread from Peer Class.
- **private String chunksToRecieve()**
  Method chunksToRecieve private to create String of not received Chunks.
  @returns a String of all chunkid separated by "^".
- **private boolean checkMoreChunkNeeded()**
  method checkMoreChunkNeeded to check if any more chunks are required or not.
  @returns true if for any chunk value in map is set to false else returns true.

Compilation Details

File Owner:
For File owner the class can be compiled using following command.

```
javac FileOwner.java
```

Running Instruction:
For running

```
java FileOwner <Server_Port> <File_name>
```

Peer:
For Peer the software can be compiled using following command.

```
Javac Peer.java
```

Running Instruction:
For running Peer:

```
java Peer <Server IP> <Server Port> <Upload Port> [Peer IP] <Download Port>
```

[] – Optional

Enviornment Details:
Java 1.7

Sample file is attached in the code for distribution among peers.
Example Command:

```
java FileOwner 5000 sample.mp4

java Peer 127.0.0.1 5000 5001 5005
java Peer 127.0.0.1 5000 5002 5001
java Peer 127.0.0.1 5000 5003 5002
java Peer 127.0.0.1 5000 5004 5003
java Peer 127.0.0.1 5000 5005 5004
```

Also implemented is feature for distributed peer over the LAN for which downloading peer IP needs to be specified which is optional. The software so developed will work across LAN for file sharing among peers. For LAN above software can be used as:

```
java Peer 127.0.0.1 5000 5001 172.31.122.12 5005.
```