Lab 1: Peer-to-Peer File System

CSE 513 - Distributed Systems
The Pennsylvania State University
Fall 2019

Submitted On:	10/11/2019	
Submitted By:	Sahithi Rampalli	(svr46)
	Gulsum Gudukbay	(gxg5138)

System Description and Protocol Specifications

Our Peer-to-Peer File System implementation uses a Java API - Java Remote Method Invocation (Java RMI) to communicate among peers and tracker. Java RMI is based on the idea of Remote Procedure Calls (RPCs). It uses the Java Remote Method Protocol (JRMP) and the transport layer protocol - TCP/IP.

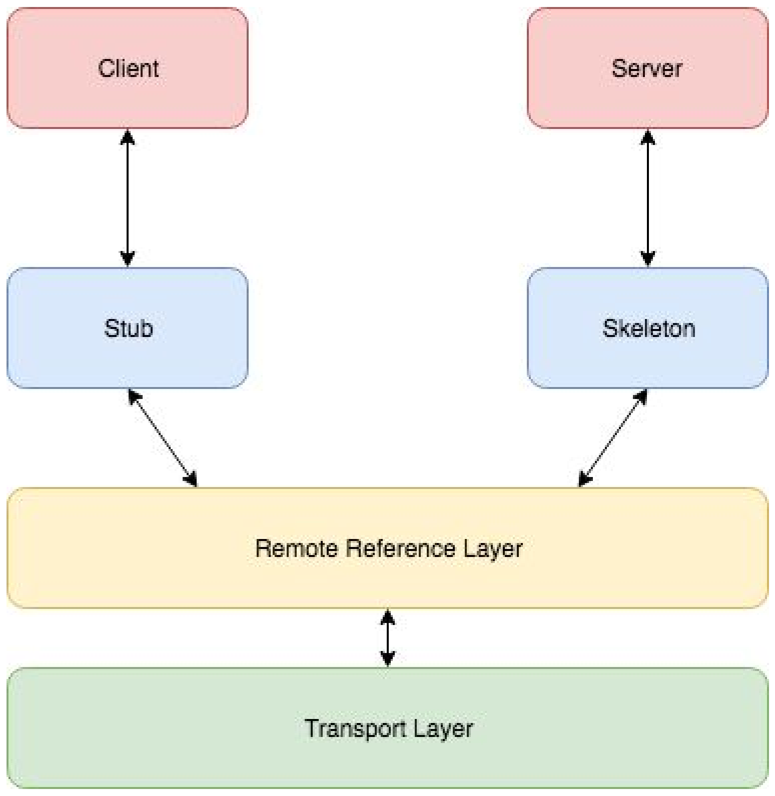


Figure 1 - Client-Server Diagram for Java RMI

As soon as a Peer/Tracker is up, it registers itself in a “rmiregistry” with a unique name. By registering itself in the rmiregistry, it makes its remote interfaces available for the rest of the world. Any Peer which would like to communicate with a Peer/Tracker can lookup the server Peer/Tracker in the remote rmiregistry, create an instance of the remote interface (called stub) and invoke the required method. On the server side, the interface is accessed (called skeleton), the method is executed and output is sent back via RMI.

Bittorrent Implementation

Functionalities of the Tracker

- **RegisterReply** - This method, registers a peer, registers the list of files and file chunks the peer has in its database.
- **FileListReply** - When a peer recently joined the network, it requests for the list of files the peers in the network have. This method reads the tracker database and sends the list of files to the peer.
- **FileLocationReply** - When a peer requires a file, it requests the tracker for the locations of each chunk of the file. The tracker reads its database and sends the list of locations for each chunk to the peer.
- **ChunkRegisterReply** - When a peer receives a new chunk, it registers the chunks in the Tracker. The tracker updates its database with the new chunk and its location.
- **LeaveReply** - When a peer is leaving the network, it signals the tracker. The tracker deletes all the information related to the peer from its database.
- **HeartBeat** - This method receives the heartbeats from the peers. If a peer crashes, it does not HeartBeat and if a peer does not heart beat for three epochs, it is removed from the network and all its data in the tracker database is deleted.
- *Note*: For every operation above, the database is updated with necessary data. This ensures no loss of information in case the tracker dies abruptly.

Functionalities of the Peer

- **RegisterRequestToServer** - When a peer is up, it registers itself in the server, sends the list of files and file chunks the peer has. The data of the files and file chunks the peer has is stored in its database. If the peer dies abruptly or when it joins the network, it still has the list of files and file chunks.
- **FileListRequest** - When a peer is up, it requests the tracker for the list of files in the network.
- **FileLocationsRequest** - When a peer wants to download a file, it requests the tracker for the locations of the chunks of the required file.
- **ChunkRegisterRequest** - When a peer has a new chunk which is not yet registered by it on the tracker, it sends the chunk information to the tracker.
- **LeaveRequest** - When a peer is about to leave, it signals the tracker to delete its data.
- **PeerFileRecv** - When a peer wants a chunk from another peer, this method is invoked to receive the chunk.
- **FileChunkReply (PeerFileSend)** - When a peer is requested a file chunk by another peer, this method is invoked to send the chunk.
- **HeartBeatHandler** - The peer has to periodically let the tracker know if it is alive. This handler runs every ten seconds to tell the tracker the peer is alive.

Program Structure Description

Java Interfaces

- ITracker - Remote Interface for the Tracker.
- IPeer - Remote Interface for a Peer.

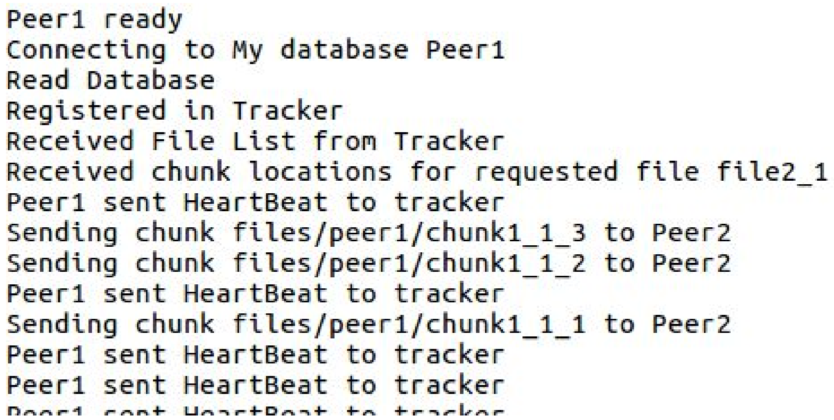
Java Classes

- DBInit - Used to initialize the databases of the tracker and peer
- Tracker - Tracker class implements ITracker remote interface. (The methods were discussed above).
- Peer - Peer class implements IPeer remote interface.
- HeartBeatHandler - Handles Heart beats of peers.

Implemented Functionalities

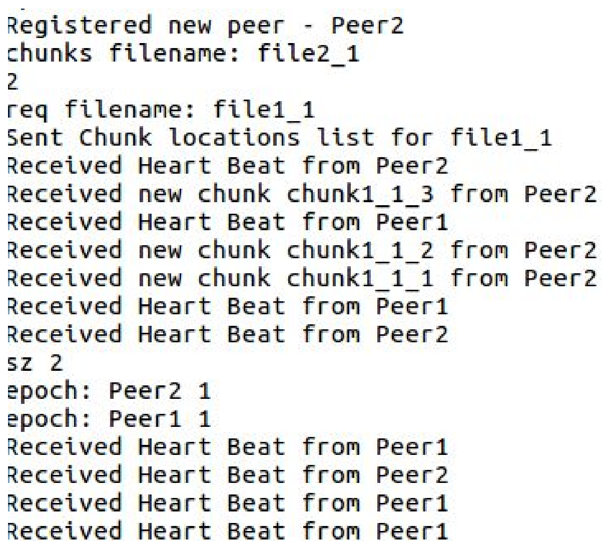
- **Multiple Connections**: The peers and tracker can handle multiple requests at a time. We use java RMI Registry which is multithreaded. It also buffers certain requests.
- **Parallel Downloading**: Our system is able to download a file from multiple peers and assemble them using the Java Thread class polymorphism.
- **Chunk Selection**: Upon receiving a chunk, the peer registers the chunk in the tracker using ChunkRegisterRequest method. We have an option to use “rarest first” for file downloading.
  - The peer may or may not choose to use the ‘**rarest first**’ algorithm for downloading. It can choose rarest first by providing “rarest” as a command line argument.
- **Chunk Download Completion:** When a peer finishes downloading a chunk when receiving it, it registers that chunk to the tracker for future references to state that it can also be a source of that chunk now.
- **Failure Tolerance:** The system **does not crash** if a peer crashes abruptly. The Tracker and a peer can **restore all the information** they had before crashing as the data is stored in database. For instance, a peer can restore all the information of the files it had before crashing and again register them in the Tracker.
  - The tracker keeps a check on each peer every three epochs. If a peer does not **heartbeat** for the last three epochs, the **tracker removes the peer** from the network and deletes all its data. When the peer restores, it can register all its data into the tracker again.
- **Incentive:** Each peer will have friends. A peer checks if the requesting node is one of its friends, and if it is, it immediately processes the request. Otherwise, the request is queued and every 15 seconds, it randomly selects a requesting peer from its waiting queue and serves the request. The peers send chunks to its friends at its highest data rate, and it chokes others by making them wait in the queue and after time t (15 seconds for our demo), the peer randomly selects another peer in its queue and optimistically unchokes it.
  - The **speeds** of each peer can be changed by specifying a delay amount from the command line argument.

Sample Outputs



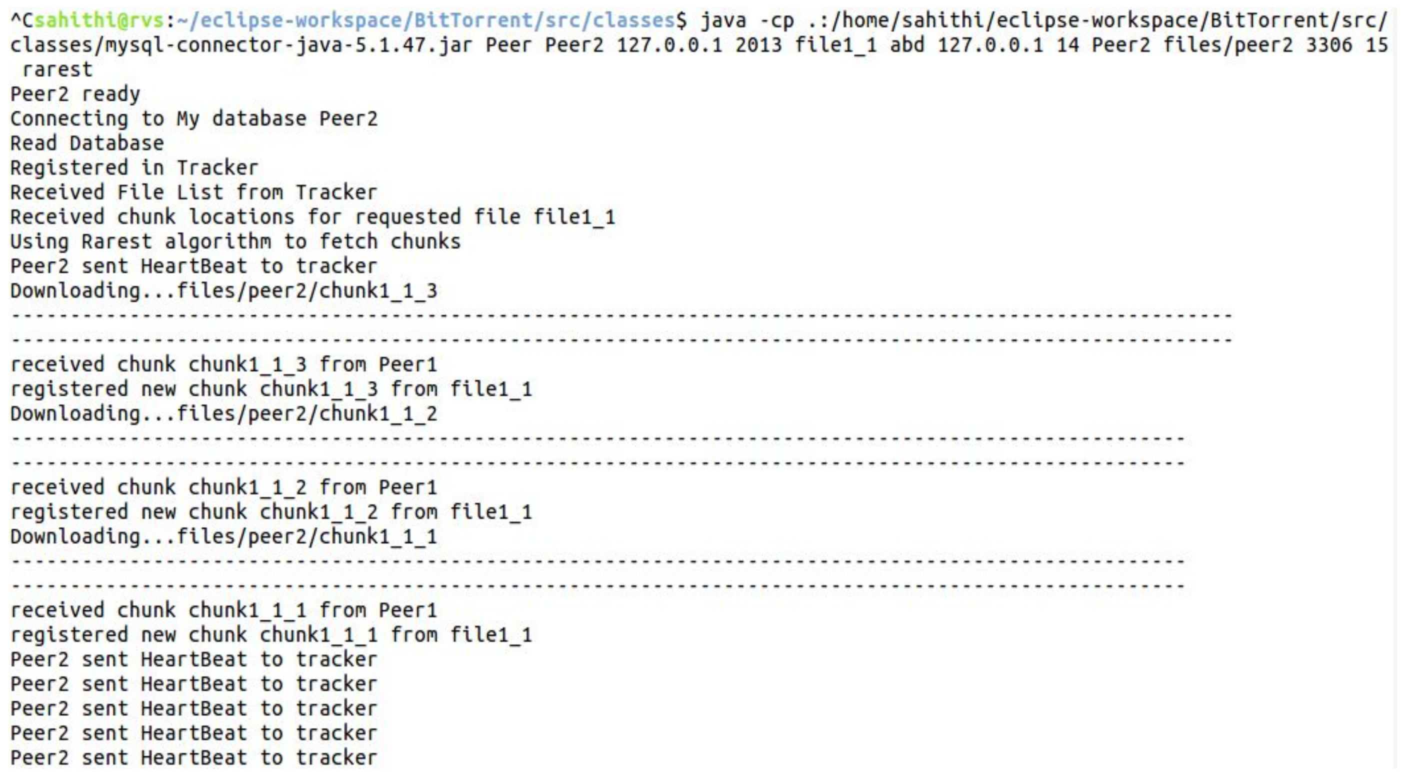
```
Peer1 ready
Connecting to My database Peer1
Read Database
Registered in Tracker
Received File List from Tracker
Received chunk locations for requested file file2_1
Peer1 sent HeartBeat to tracker
Sending chunk files/peer1/chunk1_1_3 to Peer2
Sending chunk files/peer1/chunk1_1_2 to Peer2
Peer1 sent HeartBeat to tracker
Sending chunk files/peer1/chunk1_1_1 to Peer2
Peer1 sent HeartBeat to tracker
Peer1 sent HeartBeat to tracker
Peer1 sent HeartBeat to tracker
```

Figure 2 - Peer1 sending its file chunks to Peer1. First, Peer1 declares itself as ready and connects to its own database. Peer1 then fills its data structures from its own database and registers itself in the Tracker. Then, Peer1 requests and gets the "File List" from the Tracker as seen in line 5 of the sample output. Then, it requests and receives the "chunk locations" for requested file, which was given to the command line argument as file2_1 as seen in line 6 of the sample output.
As the file operations are going on, we can see "Peer1 sent Heartbeat to tracker" lines around different lines of the output and that shows that the peer tells the tracker that it is alive and did not crash unexpectedly periodically. Then, When Peer2 requests a file from Peer1, Peer1 sends the chunks to Peer2. We can see that it is sending the chunks in lines 8, 9 and 11.



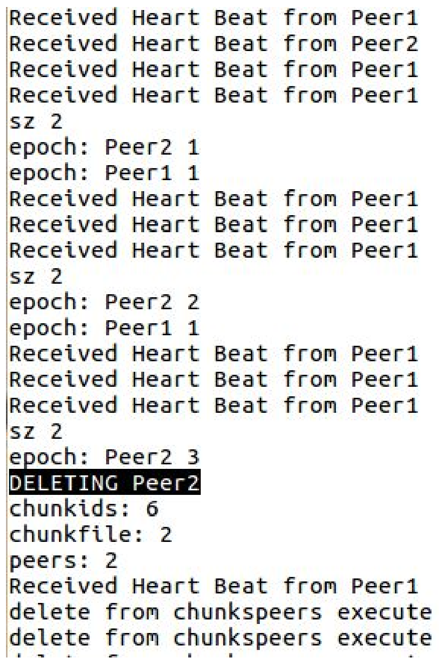
```
Registered new peer - Peer2
chunks filename: file2_1
2
req filename: file1_1
Sent Chunk locations list for file1_1
Received Heart Beat from Peer2
Received new chunk chunk1_1_3 from Peer2
Received Heart Beat from Peer1
Received new chunk chunk1_1_2 from Peer2
Received new chunk chunk1_1_1 from Peer2
Received Heart Beat from Peer1
Received Heart Beat from Peer2
sz 2
epoch: Peer2 1
epoch: Peer1 1
Received Heart Beat from Peer1
Received Heart Beat from Peer2
Received Heart Beat from Peer1
Received Heart Beat from Peer1
```

Figure 3 - This is a sample output for the Tracker. Here, we can see that Peer2 is registered to the tracker. When the peers are registered to the tracker, they are added to the Peers table of the tracker database. Then, the desired filename is printed. The chunk locations list is sent to the Peer2. Peer2 then sends its chunks to the Tracker and the chunk list is updated. Epochs are printed to see if both peers are alive.



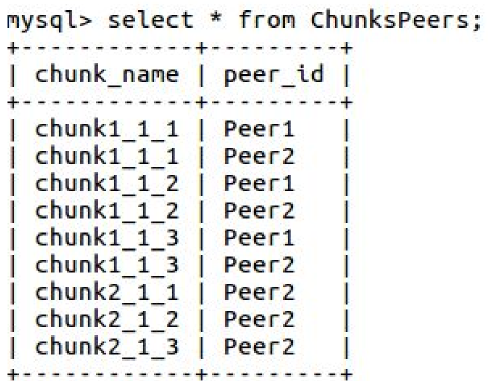
```
^Csahithi@rvs:~/eclipse-workspace/BitTorrent/src/classes$ java -cp .:/home/sahithi/eclipse-workspace/BitTorrent/src/
classes/mysql-connector-java-5.1.47.jar Peer Peer2 127.0.0.1 2013 file1_1 abd 127.0.0.1 14 Peer2 files/peer2 3306 15
 rarest
Peer2 ready
Connecting to My database Peer2
Read Database
Registered in Tracker
Received File List from Tracker
Received chunk locations for requested file file1_1
Using Rarest algorithm to fetch chunks
Peer2 sent HeartBeat to tracker
Downloading...files/peer2/chunk1_1_3
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
received chunk chunk1_1_3 from Peer1
registered new chunk chunk1_1_3 from file1_1
Downloading...files/peer2/chunk1_1_2
--------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------
received chunk chunk1_1_2 from Peer1
registered new chunk chunk1_1_2 from file1_1
Downloading...files/peer2/chunk1_1_1
--------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------
received chunk chunk1_1_1 from Peer1
registered new chunk chunk1_1_1 from file1_1
Peer2 sent HeartBeat to tracker
Peer2 sent HeartBeat to tracker
Peer2 sent HeartBeat to tracker
Peer2 sent HeartBeat to tracker
Peer2 sent HeartBeat to tracker
```

Figure 4 - Peer2 receiving chunks from Peer1 for its desired file. We can see that when receiving chunks from other peers, Peer2 uses rarest first algorithm to get them from different peers and we can see this in line 7 of the sample output. We can then see the download progress of each file chunk. There are two consecutive lines of dashes for each file. The first line indicates the full file size and the second indicates the written file size and it increases as time passes and as file writes are happening. So we can see the download progress. The output above shows the finished downloads, since we cannot put an animation to the report. After download is finished, we can see that Peer2 did not crash, because the Peer2 sends heart beats to the tracker.



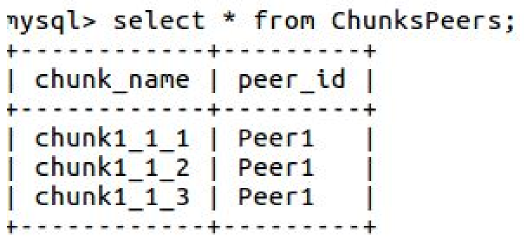
```
Received Heart Beat from Peer1
Received Heart Beat from Peer2
Received Heart Beat from Peer1
Received Heart Beat from Peer1
sz 2
epoch: Peer2 1
epoch: Peer1 1
Received Heart Beat from Peer1
Received Heart Beat from Peer1
Received Heart Beat from Peer1
sz 2
epoch: Peer2 2
epoch: Peer1 1
Received Heart Beat from Peer1
Received Heart Beat from Peer1
Received Heart Beat from Peer1
sz 2
epoch: Peer2 3
DELETING Peer2
chunkids: 6
chunkfile: 2
peers: 2
Received Heart Beat from Peer1
delete from chunkspeers execute
delete from chunkspeers execute
```

Figure 5 - Peer2 crashing abruptly. The output above shows that Peer2 crashes abruptly. We understand that it crashed by counting epochs and if it did not send a heartbeat for 2 epochs, we count it as a crashed Peer, so we call leaveReply and delete peer and its contents from the data structures and the database.



```
mysql> select * from ChunksPeers;
+------------+---------+
| chunk_name | peer_id |
+------------+---------+
| chunk1_1_1 | Peer1   |
| chunk1_1_1 | Peer2   |
| chunk1_1_2 | Peer1   |
| chunk1_1_2 | Peer2   |
| chunk1_1_3 | Peer1   |
| chunk1_1_3 | Peer2   |
| chunk2_1_1 | Peer2   |
| chunk2_1_2 | Peer2   |
| chunk2_1_3 | Peer2   |
+------------+---------+
```

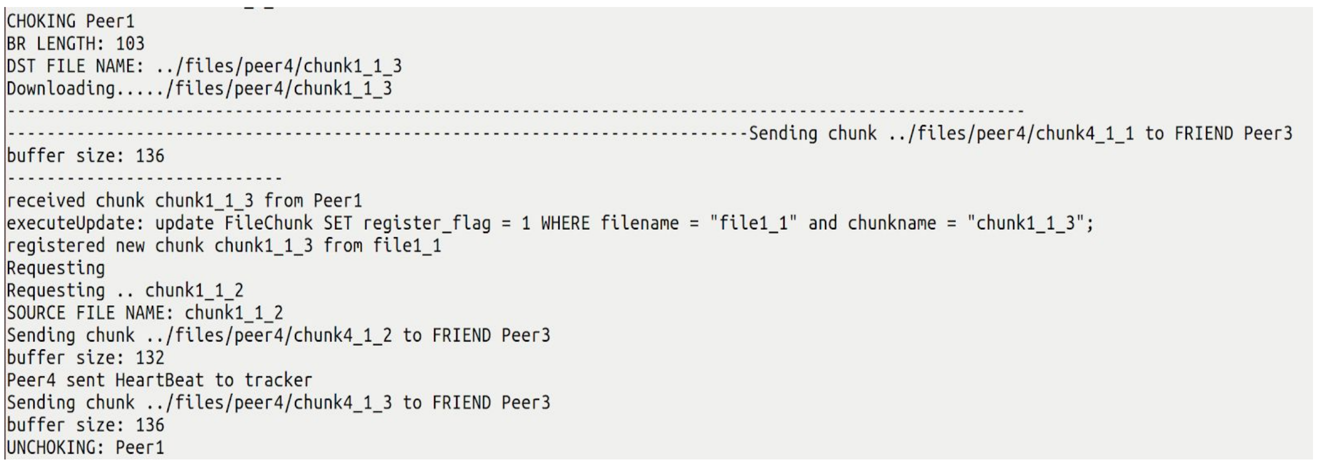
Figure 6 (a) - Database table before LeaveReply is called.



```
mysql> select * from ChunksPeers;
+------------+---------+
| chunk_name | peer_id |
+------------+---------+
| chunk1_1_1 | Peer1   |
| chunk1_1_2 | Peer1   |
| chunk1_1_3 | Peer1   |
+------------+---------+
```

Figure 6 (b) - Database table after LeaveReply is called.

Figure 6 - We can see from the output that after Peer2 has crashed abruptly, the files of Peer2 is deleted from the database.



```
CHOKING Peer1
BR LENGTH: 103
DST FILE NAME: ../files/peer4/chunk1_1_3
Downloading...../files/peer4/chunk1_1_3
--------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------Sending chunk ../files/peer4/chunk4_1_1 to FRIEND Peer3
buffer size: 136
----------------------------
received chunk chunk1_1_3 from Peer1
executeUpdate: update FileChunk SET register_flag = 1 WHERE filename = "file1_1" and chunkname = "chunk1_1_3";
registered new chunk chunk1_1_3 from file1_1
Requesting
Requesting .. chunk1_1_2
SOURCE FILE NAME: chunk1_1_2
Sending chunk ../files/peer4/chunk4_1_2 to FRIEND Peer3
buffer size: 132
Peer4 sent HeartBeat to tracker
Sending chunk ../files/peer4/chunk4_1_3 to FRIEND Peer3
buffer size: 136
UNCHOKING: Peer1
```

Figure 7 - Here Peer4 chokes Peer1 until it randomly selects it and optimistically unchokes it. However, it sends chunks to Friend Peer3 in maximum speed.

Compile and Run Instructions

Software Requirements

- MySql version 5.0 or above that has database <tracker_database_name> and <peer_database_name> for each peer in it.
- Java version 8.0 or above
- Installation of Java JDBC for mysql and then link the path to jdbc to the jar of JDBC:

```
CLASSPATH=$CLASSPATH:/usr/share/java/mysql.jar
export CLASSPATH
```

Compile Instructions

1. cd <source_code_directory>
2. mkdir classes
3. javac -d classes *.java

Run Instructions

Tracker

1. cd classes
2. java DBInit 1 trial
3. java Tracker trial

Peer

1. cd classes
2. Java DBInit 2 <Peer DB name> <directory name for peer>
3. Java Peer <peer_id> <peer_ip> <peer_port> <requested_filename> <requested_filename> <tracker_ip> <tracker_port> <peer_database_name> <peer_directory_for_sharing> 3306 <peer_heartbeat_period> <"rarest" for rarest first algorithm, random string for otherwise> <peer id of friend 1> <peer id of friend 2>
4. Example:
   - java DBInit 2 Peer1 ../files/peer1/
   - java DBInit 2 Peer2 ../files/peer2/
   - java DBInit 2 Peer3 ../files/peer3/
   - java DBInit 2 Peer4 ../files/peer4/
   - java Peer Peer1 127.0.0.1 2011 file4_1 abd 127.0.0.1 2014 Peer1 ../files/peer1 3306 10 rarest Peer2 Peer3
   - java Peer Peer2 127.0.0.1 2013 file3_1 abd 127.0.0.1 2014 Peer2 ../files/peer2 3306 15 rarest Peer1 Peer4
   - java Peer Peer3 127.0.0.1 2012 file4_1 abd 127.0.0.1 2014 Peer3 ../files/peer3 3306 20 rarest Peer1 Peer2
   - java Peer Peer4 127.0.0.1 2015 file1_1 abd 127.0.0.1 2014 Peer5 ../files/peer4 3306 20 rarest Peer2 Peer3