**Texas Tech University**

18

**Programming Assignment #2**

**Peer to Peer – An Implementation**

**Guillermo Rodriguez**

# 1.0 Introduction

This development work effort was performed with the Java development programming language (<https://www.oracle.com/java/>). The purpose of the development effort was to create a peer to peer (P2P) file sharing solution that would allow an individual client or peer in this case to contact a central repository and obtain a listing of sources where a desired artifact could be extracted from. In this endeavor the artifact was a simple text file with simple data in it. As part of the requirement each of the individual files that could be transmitted was of a distinct size for each peer instance.

Each of the separate applications – peer and server where developed with the Java programming language. The communication mechanism that was chosen for the data transfer was sockets whereby a peer would create a connection to the server and receive a specifically formatted response from the server. What was needed to be created was a protocol of communication from peer to server as is the case in all information exchange endeavors such as TCP, HTTP, or XML. It is a fundamental necessity to be able to define the mechanism of communication between distinct nodes and to this endeavor one such mechanism was defined and utilized. While simple in nature in this implementation it does highlight a necessity of software development – communication must adhere to a strict protocol or mode of operation even in simple cases as is presented in this development effort.

Each of the applications – peer and server ran under their own weight, i.e. implementation in Java and were executed from the command line. Once both instances of the applications were functional then it was possible to begin to communicate between the peer and the server as well as in between peers for the exchanging of data files.

This implementation required the use of simple data constructs for implementation purposes such as flat files to keep track of logged on users as well as data files to keep track of each client’s data store, i.e. file repository. In an enterprise implementation this would have been changed to a physical data store construct such as a MySQL database or a MongoDB instance. What is highlighted here is that while some of the underlying architectural artifacts are distinct in this implementation as opposed to an enterprise implementation the scalability factor is there as should always be the case when considering a well sought design. Technological change or shift is an undergoing process in the development effort, but what should be fundamental and constant is the generalizations of solutions that can evolve over time without burdening the business with a large technological debt as to do so in the current development landscape could be disastrous to a business.

What proceeds in this document is a formal architectural definition of the project as well as a formal discourse into the peer and server implementation in Java along with the test cases that were created and validated.

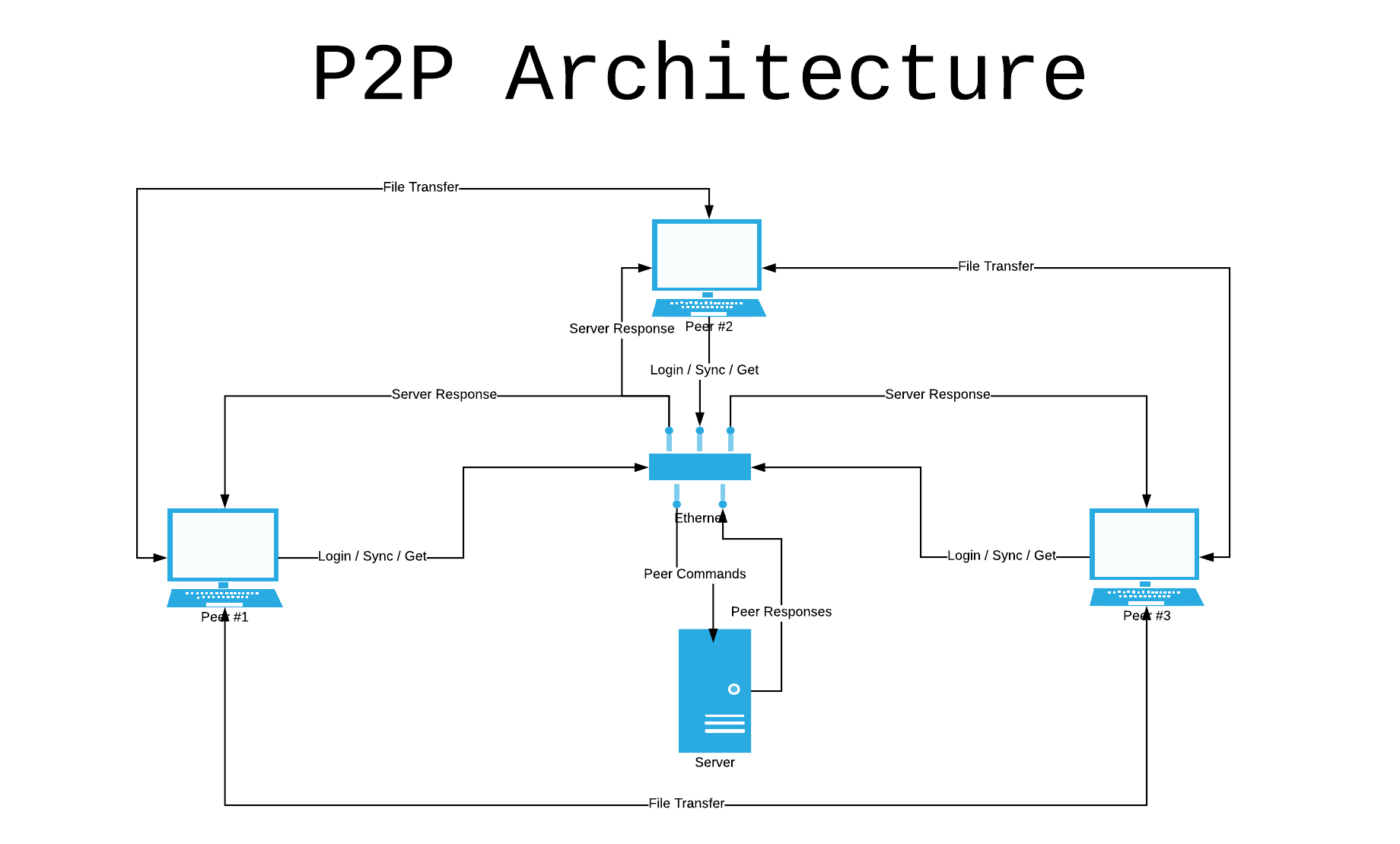
# 2.0 Architecture

The architectural framework of this project was set forth in the requirements where it was mandated that there needed to be a peer and server implementation in some programming language. To this goal what was chosen as the development platform was the Java programming language given its rich technological utilities for network programming such as sockets. Given that the implementation was needed to be executed in the Linux environment it automatically excluded development platforms such as C#; and while .NET core could have been an option under Linux the Microsoft stack was deferred given the native support in Linux for Java.

The development effort took on the architectural layout given below, where the diagram depicts three peer objects and one central node that functions as the server. The purpose of the peer nodes was multifold and they were as follows:

* Register with the server. This allowed the server to maintain a running list of available nodes to use as sources for needed files.
* Transmit a list of available documents to the central server. This allowed the central server node to keep track from where a possible file could be transmitted to.
* Get a peer node for data transmission. This allowed the central server to respond back with a list of possible nodes to obtain a needed file from.
* Transmit data files. Send needed files to a peer given a request for a file from the peer.

The peer node acted as both a client and server implementation as it needed to communicate status detail to the central server node, but also needed to serve content to peers when requested. In a peer to peer (P2P) implementation each peer node is both source and sink. This implementation highlighted this architectural construct by way of an implementation at the peer nodes by way of sockets that either accepted connection requests or through the communication of information through sockets to the corresponding nodes, i.e. server or peer.



# 3.0 Peer

The peer nodes where the functionally most complex component of the implementation as each node needed to not only provide an outgoing socket connection, but also required the component to accept connections. The peer nodes acted in a bidirectional manner as opposed to the server node that simply accepted and responded to connection request.

The peer nodes where invoked by way of the peer command in Java, i.e. java peer. The physical command to invoke the peer was as follows:

java peer [integer number]

The [integer number] value was a numeric designation that served the purpose of acting as the peer id. This peer id was the mechanism by which the central node kept track of online connections, i.e. nodes that could respond back with a file request.

Communication between the peer and the server followed a specific protocol that was of the following forms:

1. Login:[client id]:[ip address]-[port]
   1. Client Id – Unique identifier
   2. IP Address – localhost for our purposes
   3. Port - Port number to accept connections on
2. Sync:[client id]:[file list]
   1. Client Id – Unique identifier
   2. File List – Comma separated file list of documents that exist at the client node
3. Get:[client id]:[file name]
   1. Client Id – Unique identifier
   2. File Name – Name of a file to download from some peer

Communication between peers also functioned according to a specific guideline and it was as follows:

1. Get:[file name]
   1. File Name – The name of the file whose contents were being request

Once a particular node underwent a file download operation then the system state was brought to homeostasis once again by invoking the Sync command with the server node.

# 4.0 Server

The server node in this configuration essentially served the purpose of a state machine, where the nodes purpose was to keep a running state of the system. The server node was responsible for keeping track of which data files existed at the individual peer nodes as well as to keep track of the peers that were logged in. The configuration would be similar to a large public offering system as the central node would need to provide a running account of possible nodes and their IP address from where a resource could be extracted from.

The server configuration used a users.txt file that was located under the database folder of the root node to keep track of all logged in users and their corresponding IP address with port. This IP address with port combination would serve the purpose of letting a peer node know where it could communicate to in order to download a file. The configuration of the users.txt file format is given below.

* [User Id],[IP Address]-[Port]
  + User Id – Unique peer identifier
  + IP Address – Networking address
  + Port – Port number to communicate on

The server configuration kept track of peer content through a file with the client id as its name. The file was in text format and simply contained entries listing each of the files that existed at the peer node. This file list was created by way of the Sync command and was deleted and created upon each Sync command. The file may be found under the files directory from the root node of the implementation.

The server instance was launched by way of a command line that took the form of:

java server

# 5.0 Test Cases

The testing artifacts may be found under the client id folder in a folder named log which has the name log.log. This file contains the transactional detail for the communication between the client and server node as well as the transactional detail that resulted as a consequence of the communication between the peers. It was through these individual artifacts that the overall functionality of the application was verified and deemed to be working. You may find the latest build and execution sequence in its entirety via the Github repository at [www.github.com/guillermorodriguez/File-Sharing](http://www.github.com/guillermorodriguez/File-Sharing).

The log files with the transactional detail contain information about the execution of each command sequence, i.e. Login/Sync/Get and the response from the individual peer or server instance.

# 6.0 Conclusion

This project proved to be a good exercise into the architecture of a peer to peer implementation and while some of the artifacts proved to be simple in nature it did lay the ground work for the understanding of how a system may be created with sockets to communicate between disbursed components. The implementation was performed in the Java programming language and as such also allowed for the exposure of how a current development platform may be utilized to create a networked solution for communication across silos.

With the aid of simple data structures and high level network programming interfaces it was completely feasible to create a system that embodied the same architecturally relevant components as a full enterprise solution thus allowing for a very good hands on learning exercise that serves the part of filling in a picture of distributed systems and their underlying context.