**Texas Tech University**

18

**Programming Assignment #3**

**Peer to Peer – An Implementation**

**Guillermo Rodriguez**

# 1.0 Introduction

This development work effort was performed with the Python development programming language (https://www.python.org/). 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 fellow peers to perform a query for a needed file. 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 or peers where developed with the Python programming language. The communication mechanism that was chosen for the data transfer was sockets whereby a peer would create a connection to a fellow peer with a query request and receive a specifically formatted response from the fellow client. What was needed to be created was a protocol of communication from peer to peer 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 peer applications executed under their own weight, i.e. implementation in Python and were executed from the command line. Once multiple instances of the applications were functional then it was possible to begin to communicate between the peers for the querying of needed information.

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 implementations in Python along with the test cases that were created and validated.

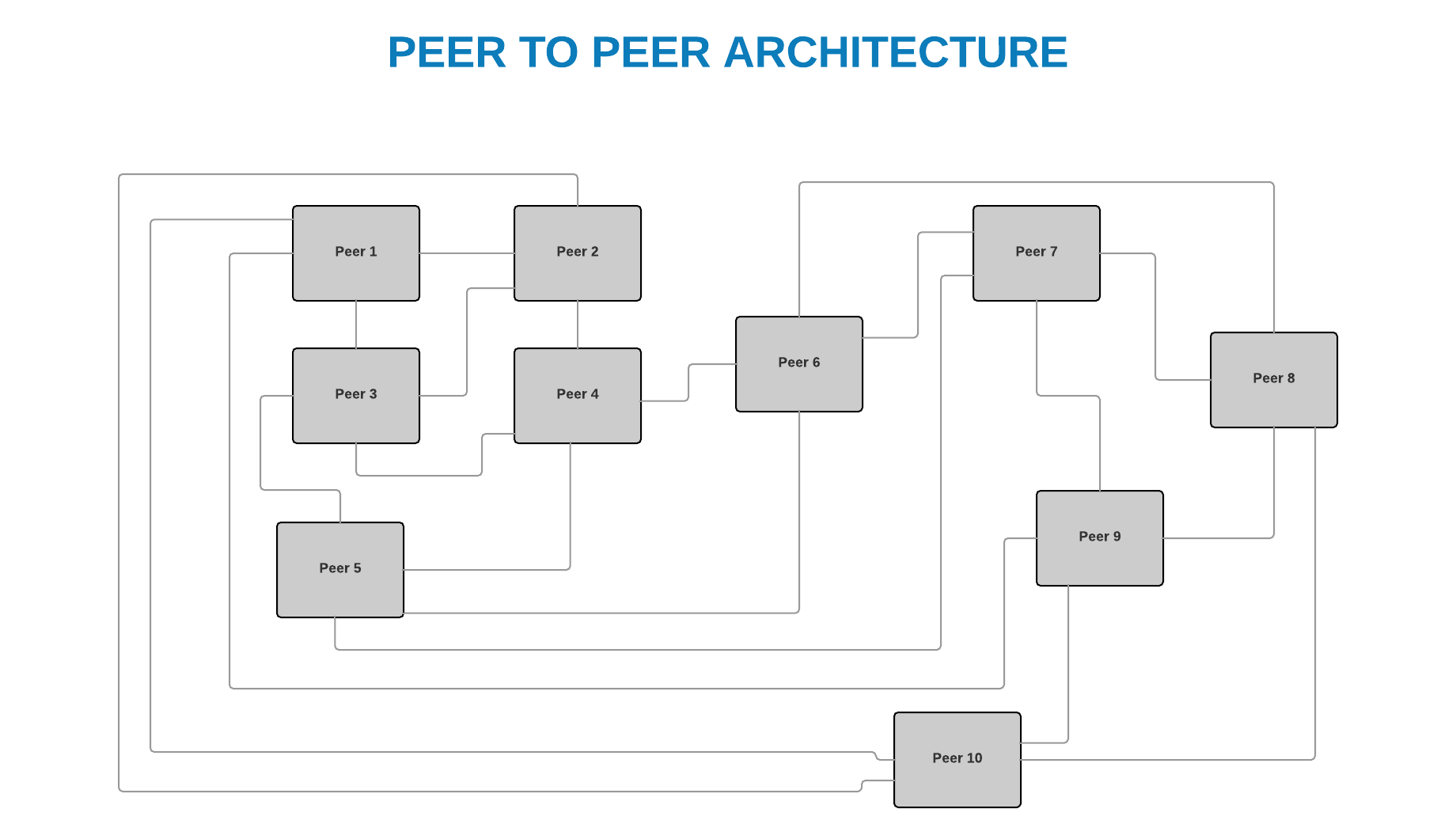
# 2.0 Architecture

The architectural framework of this project was set forth in the requirements where it was mandated that there needed to be peer to peer communication in some programming language. To this goal what was chosen as the development platform was the Python 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 Python.

The development effort took on the architectural layout given below, where the diagram depicts ten peer objects that function as both client and server instance. The purpose of the peer nodes was multifold and they were as follows:

* Upon instantiation invoked a server and client interface component that were multithreaded.
* Send a query request to peer nodes that in turn would either respond with a valid file instance or would pass the query to their respective connected nodes.
* Peer nodes were connected in an N+2 manner, where two recipients were attached to each peer node.
* Transmit data files. Send needed files to a peer given a request for a file from the peer or pass the message along to a fellow peer to respond.

The peer node acted as both a client and server implementation as it needed to communicate status detail to its connected peer nodes. The communication mechanism allowed was such that a ring formed between nodes. At the 9th node, it connected to the 10th node and then back to the first node. The tenth node connected to the 1st and 2nd nodes, closing the loop. Communication between nodes happened through socket connections.



# 3.0 Peer

The peer nodes acted as both client and server. Each peer connected to fellow peers by way of a definition file with the same name as specified as it identifier number. Once a query term was issued by the user through the command line interface, this query was forwarded to fellow peers in order to be answered. If the answer could not be determined then the query was forwarded to destination nodes of fellow peers and down the line until a communication threshold was met. The communication threshold was set to 2 for this exercise, but in theory could have left as open.

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

python peer.py [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 peer nodes were identified and the mechanism by which the peer configuration file was identified when routing was needed.

Communication between the peers followed a specific protocol that was of the following form:

1. Query:[File Sought], [Threshold]
   1. File Sought – file to search for
   2. Threshold – Number of peers to pass the request to if it could not be answered
2. Expired: [File]
   1. File – File name that is outdated

# 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’s nodes as well as the transactional detail that resulted as a consequence of the communication. 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/peer](http://www.github.com/guillermorodriguez/peer).

The log files with the transactional detail contain information about the execution of each command sequence between the individual peer nodes.

# 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 Python 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.

The reason why the Python programming language was chosen was for its flexibility with development and as such it proved to be just the case. The language allowed for threading of class objects with ease. The development platform proved to be more than adequate to solving the problem frame and it may be even argued proved to show why the programming language is being chosen over other development constructs. In true Python spirit the code base proved to be small and while still allowing for the body of work to be completed. The same functionality in a separate programming context like Java for example would have proven to have taken up more lines of code. This simplicity for implementation is significant as doing so only highlights why its adoption rate is high and why professionally here in the bay area I am seeing a shift away from other languages to Python. One only need to look at the Django framework to understand the ease with which Python development work can occur for the web for example.