==========

DOCUMENTATION

==========

==========

I - Introduction

==========

This milestone of the project makes the server and client both send message using ASN.1 encoding. The server and client programs are also combined into one, with a single program starting both the client and the server in separate threads. The final change made in this milestone is to ensure that the server will not close a TCP connection until the client closes the connection or the connection times out, and will be prepared to receive multiple messages over that single TCP connection. This document will describe the changes made to the server and client to implement those changes.

==========

II - Technologies Used

==========

The project is implemented in Java, and relies on the Apache Commons CLI library for parsing command line options. We opted to use the file system rather than a database library for saving information.

We introduced open source code to produce the hash function for gossip messages in the client program. Specifically, the SHA256 class was found on github at the following link:
https://github.com/BaconRemovalUnit/SHA256/blob/master/src/SHA256.java

We also used the provided ASN.1 library, DirectDemocracyP2P, for using ASN.1 in Java found on the class website. This library is on gtihub at the following link:
https://github.com/ddp2p/DDP2P/tree/master/src/java/net/ddp2p/ASN1

Generating the user manual for the project will require javadoc. The specific script for doing so is "generateDocs.sh"

==========

III - Architecture

==========

i) Structure

The structure of the program changed significantly due to the combing of the client and server programs into one program. The program is now started by running the Master java class. The Master class takes in all command line arguments for both the client and the server, utilizing the apache commons CLI parsing library, and starts the server in one thread and the client in another thread. The specific structure of both the server and the client is still relatively unchanged. The server starts in the Main class, which starts the TCPServer and UDPServer threads, each of which accepts a connection and passes that connection to the TCPConnectionHandler or UDPConnectionHandler

class respectively to handle the connection. Both of those classes extend the GossipServer class and so make use of much of the functionality provided by that class to process the messages received and interact with the data files. The networking aspect is still handled by the connection handler classes. Both connection handler classes also make use of utilities contained in the AppUtils class, which contains various utilities used by multiple classes in the project, and the code that parses the ASN.1 message with a decoder is in the AppUtils class, along with the code to continually read from the TCP socket to ensure that an entire message has been received over TCP. Two main datatype classes still exist, Peer.java and Gossip.java, for easy formatting of the data associated with peers and gossip messages. Those two classes are where the bulk of the ASN.1 encoding is contained, as they both have a function that simply returns the byte array containing the ASN.1 formatted data that can be sent over the network. The client part of the program is also formatted just like it was previously, with the ClientApp class handling most of the work and relying on an object of type ClientConnection, a datatype class that both the TCPClient and UDPClient classes extend to allow polymorphism to direct the networking aspect of the client to the appropriate class based on the connection type (UDP or TCP). The ASN.1 related changes made to the client part of the program are very small since the client part also relies on the same Peer.java and Gossip.java datatype classes, which were already set up to return the proper ASN.1 encoded byte array, ready to be sent over the network.

ii)Design Choices

The design choices that had to be made for this milestone of the project are mostly related to the change to use ASN.1 to encode messages to be sent over the network. The design choices made were in relation to where to implement the encoding and decoding of ASN.1 messages. The gossip and peer messages are encoded in the Peer.java and Gossip.java classes. The peersQuerry message is encoded right before it is sent, in the place that it is sent. This is because it takes only two lines of code to create the encoding for that message since it is just the null encoder. The peersAnswer message is also encoded in GossipServer.java directly before it is sent because it is only ever sent from that one location and so it only ever needs to be encoded in that one location. The decoding of messages for the server is done by the parseReadyDecoder method in the AppUtils class. That utility method is used by both the TCP and UDP server threads once they have acquired a full message into a decoder via their respective network types. The decoding of the peerAnswer message by the client, the only message the client should ever receive, is handled within the TCPClient or UDPClient classes directly following the network-type specific retrieval of the message from the network into the decoder object.

iii)Data File Format Documentation

The format of the data files that we use is still the exact same as it was before. The server will generate two simple text files, one for gossip messages and one for known peers, without that dynamically generated number, simply called "gossip.txt" and "peers.txt", and it will start with

whatever data those text files hold before the server was started. The specifications again are detailed below:

gosssip.txt:

Because the gossip messages are forwarded to known peers in the exact format that they are received in, there is no need to change the way that the data is arranged for storage in the gossip data file. Each unique gossip message received is stored in the gossip data file with the format:

<encoding>:<dateTime>:<message>%

The first field, GOSSIP, does not need to be included since all data in the gossip data file is already known to be about gossip messages.

peersFile.txt:

Each peer only requires one line since the newline character cannot appear in the information about a peer, so the format is simple. Each line in the peers data file contains information about a single peer in the format:

<name> <port> <ip>

The three fields can be separated by any amount of whitespace, but must always be separated by at least some whitespace so that they can be read properly.

==========

IV - User Manual

==========

The user manual can be generated through javadoc by using the script "generateDocs.sh" I highly recommend making auto-generated user manuals a requirement in future classes, as it leads to more easily readable and accessible code. It is a very useful skill to have in general. Javadoc is not available for C++ users, but Doxygen does basically the same thing and is language-agnostic.

==========

V - Conclusion

==========

In conclusion, the server and client programs have successfully been merged into one master program that starts both programs as separate threads. The entire program has been changed to use ASN.1 format for encoding and decoding messages to send over the network. The TCP part of the server has been changed to now let TCP connections persist, closing only when the client closes the connection or when the connection time out after a time of twenty seconds. It meets all given specifications and is a simplified version of the Gossip P2P system.

==========

VI - References

==========

This project uses command line option parsing code adapted from Dmitry Kulakov's example provided during his in-class presentation. The home page of the parsing code is: http://commons.apache.org/proper/commons-cli/

DirectDemocracyP2P ASN.1 Java Library:

https://github.com/ddp2p/DDP2P/tree/master/src/java/net/ddp2p/ASN1