Project1 Executive Summary

Assignment Overview

The major purpose of project 1 is to understand the computer networking basics by practicing socket programming. Through the implementation process, we were able to understand the client-server design pattern and its inner communication: (1) how data flows from client to server(how client's requests are sended to server) and (2) how data flows back from server to client(how server sends back corresponding responses to client). Furthermore, we discovered the differences between TCP and UDP in theory in the first week lecture, course materials and homework 1, but implementing this project 1 is to testify the differences more in practical programming as well as learn about their potential application scenarios in the real world.
In addition, practices contain:

1. **Design and programming also include:**

- Socket and I/O stream classes for networks communication
- Protocols design for different requests
- HashMap data structure for handling key value pairs data storage, which is a super common and powerful data form in networking and distributed systems
- Logger class for handle both request and response logs in a human-readable fashion
- Timestamp class for retrieving millisecond precision for logs and time-out handling for clients connection
- Java object-oriented design

2. **Learning to use new tools:**

- Docker(learn to use a container, pack the application to fit different Operating Systems aligns with the idea of building distributed systems)
- Shell scripts

Technical Impression

Some **clarifications** on the Project1 implementation:

1. **OOD Design structure:**

The **client** package includes:

- Client objects and their operations with related files: interface *IClient* , abstract class *AbstractClient*, concrete classes *TCPClient* and *UDPClient*
- Logs print out in human-readable format with timestamps of current system time: *ClientLogger*
- Client App/Controller with the main method: *ClientController*

The **server** package includes:
- Server objects and their operations with related files: interface *IServer*, abstract class *AbstractServer*, concrete classes *TCPServer* and *UDPServer*
- Logs print out in human-readable format with timestamps of current system time: *ServerLogger*
- KeyValue store and operations with related files: interface *KeyValue*, class *KeyValueStore*
- Server App/Controller with the main method: *ServerController*

2. Protocol Design

Currently for handling response/request parsing, since there is no specific requirement for the project, I used a super rough design - performing regular expressions with strings to deal with requests by splitting them with space. As for responses, I used ";" as a separator to handle code and response repsectively. This was highly based on the assumption that we only include a limited type of requests, code and reponses.

3. Implementing details(more can be found in comments and documentation in code)
- For the put method, Putting in an already-in-the-store key with a new value will override the old value.
- To create a "running-forever" TCP server, a nested loop was applied so that when the user prompts "exit" from the client, the server will just break from the inner loop, but never stop the outer loop, which can continue to hang there and listen to the port and waiting for clients connection.

Improvements for future works:

1. Scalability and Modularity consideration for Protocol Design. Instead of handling strings and performing regular expressions, we can create classes for request and response to handle them specifically. Thus in future implementations if we need to handle a larger number/type of requests and responses, we have more flexibility on customizing the design.

2. Scalability consideration for client-server pattern. It would be powerful if a single server can handle multiple requests from multiple clients - this aligns with the idea of a scalable distributed system.