CSCI651.03 Project #1

The goal of this project is to familiarize with the socket programming API and design a protocol with a client, server, and proxy server applications. The project is expected to be completed on an individual basis with your own proprietary protocol; there is no requirement to interoperate with another classmate's application.

Application Requirements:

The application shall provide the user the ability to query a "time" server and provide the user the current configured "time" expressed in either Unix epoch time or Coordinated Universal Time (UTC) formats. A multi-threaded proxy server will sit between clients and the origin time server to handle simultaneous requests from multiple clients. A client with the appropriate credentials shall be able to modify the "time" on the server.

The user should be presented the option to use either UDP or TCP to communicate through the network when using the client application. A proxy server should be configurable to either proxy using the client's chosen transport protocol, use only UDP, or use only TCP to connect to the origin server.

The user shall receive a report of the delay to receive an answer. The report shall provide a breakdown of the delay for each hop through a proxy server (if any) in the network.

System Requirements:

Application will be written using Java, C, or C++ programming language. The application should run on Ubuntu 14.04.3 LTS.

Deliverables:

- Documentation on the application protocol messages, formats, error codes, etc. including example session flows for success and failure cases
- Application source code and associated Make files or pre-compiled binaries. Source code should be well organized and clearly documented using an acceptable standard (such as Doxygen comments). Additional documentation such as class diagrams, sequence diagrams, etc. as needed to adequately communicate the design.

Hints:

Consider first the application protocol: What messages does the client need to send to the server? Server to client? What information does each message need to contain and how will it be represented?

Use an iterative approach to implementation: First implement a basic client-server model using UDP, then add TCP, add RTT calculation, etc. Finally, implement the proxy server.

Review course material in the text book and study well known command line applications such as `ping`, `tracert`, and `iperf`.

Invoking the Application

Rather than distinct binaries, a single binary will present the user with the modes of operation and associated options. Ignoring any language specific context, such as 'java', the application shall be invoked from the command line as follows.

`tsapp -{c,s,p} [options] [server address] port [2nd port]`

Command line options: ({} are required, [] are optional based on use case/application)

`-c`: run as client

`-s`: run as server

`-p`: run as proxy

`-u`: use UDP. *client & proxy server applications*

`-t`: use TCP. *client & proxy server applications*

`-z`: use UTC time. *client applications.*

`-T <time>`: set server time. *client & server applications.*

`--user <name>`: credentials to use. *client & server applications*

`--pass <password>`: credentials to use. *client & server applications.*

`-n <#>`: number of consecutive times to query the server. *client applications.*

`server address`: address of server to connect to. *client & proxy server applications*.

`port`: primary connection port. *server & proxy applications use this as UDP receiving port. client uses this as destination port for both UDP/TCP based on* –u*.*

`2nd port`: alt. connection port. *server & proxy use this as TCP listening port.*