Speed Jester

Design Document

Prepared by:

Aaron Monahan, Eduardo Souto, Filipe Caldas

Team: Wild Mutants

December 16th, 2015

Table of Contents

1. INTRODUCTION

1.1. Purpose

The purpose of this document is to outline the design decisions in our project. Included is the team responsibilities, objectives and our functional and nonfunctional requirements. Additionally, we will cover Speed Jester's design overview and cover topics such as:

- Data flow and design
- Design constraints and restrictions
- User interface design
- Design Considerations

1.2. Objectives

Speed Jester will be tracking the download/upload rate of data on a network device which the user's computer is connected to. A user interface will then display the results of the speed test.
The Graphical User Interface will be interactive and will show the progress of the test and will allow the user to restart and stop each test. These objectives are spelled out in further detail in the functional requirements portion of this document (see 4.)

1.3. Definitions

- Packet - is a unit of data made into a single package that travels along a given network path.
- k/M/G - kilobyte = 1,024 bytes, megabyte = 1,024 kilobytes, gigabyte = 1,024 megabytes. This unit is the data transfer rate.
- Socket - one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to. An endpoint is a combination of an IP address and a port number.

- JFrame - Java's Graphical User Interfacing tool

2. TEAM MEMBERS AND RESPONSIBILITIES

2.1. Aaron Monahan

Functional Requirements:

- 4.1
- 4.3
- 4.4
- 4.6

2.2. Eduardo Souto da Silva

Functional Requirements:

- 4.1
- 4.3
- 4.6
- 4.9

2.3. Filipe Caldas

Functional Requirements:

- 4.2
- 4.4
- 4.5
- 4.7
- 4.8

3. DESIGN OVERVIEW

3.1. Introduction and Functionality

This design overview will describe and help illustrate how Speed Jester functions. The application works through a client-server network connection. The connection is established via network sockets as shown in Figure 1.

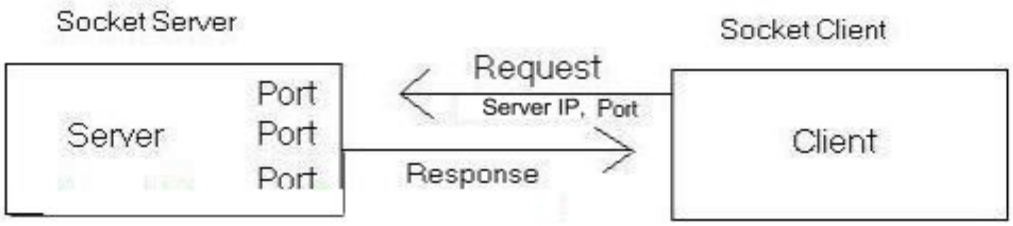

Figure 1: Simple Client - Server socket connection.

The client machine will request a connection with a server machine *openConnection()* which should be connected in the same network the client is connected to. The server has an open socket and is waiting to receive the client's request. The client will specify the server ip address and port to connect on.

Once the connection is established, we begin our speed tests. The tests consist of two parts: first, the network download speed *downloadTest()* and secondly the network upload speed. The user interface displays the current download/upload rates in kbps/Mbps/Gbps units.

For the duration of the test, we first measure the download transferring rate by receiving data sent by the server. Data will be sent through the socket and received by the client. Depending on the number of data chunks received during the test, we can calculate the rate by dividing it by the time received.

Then, for the second portion of the test, we measure the upload speed *uploadTest()* of data sent to the server by the client. This is done by essentially swapping the roles of the client and server. The client will now send data to the server and reply to the client how quickly that data was received by the server.

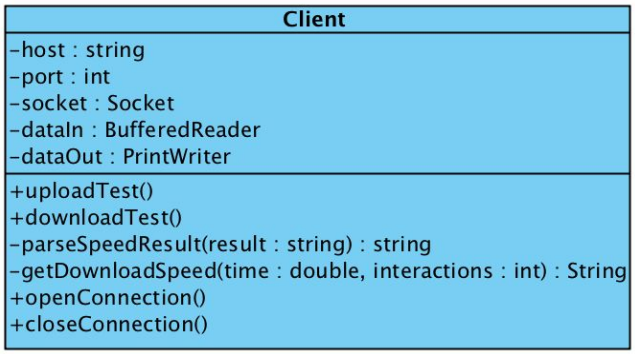

Figure 2: Client class diagram.

3.2. Design Considerations

Speed Jester is written entirely in Java. The network connections are established using the "java.net.Socket" library. This has proven to be an effective tool for our application. Although we have not worked with sockets in java, it was not very difficult to implement since syntax and functionality are consistent between most languages. Java JFrame was used to create the interface for the application. There were not many difficulties in getting this to function properly. Our main function creates the initial frame and responds according to which button the user presses and runs either the client or the server.

Although our application would work through remote server, we did not get to testing it. We did, however, test on LAN networks and it functions properly. The only thing we would need to run on a remote server would be a static ip address.

4. FUNCTIONAL REQUIREMENTS

4.1. The program will allow a user to input the server IP address and port number.

4.2. The user shall be prompted to start the program to test download/upload speed.

4.3. The program shall display results of each test in the user interface.

4.4. While the tests are running, the program will display live updates on transfer rate.

4.5. The program shall display error messages to the user in case of failure.

4.6. The test will terminate automatically after a 20 seconds of testing.

4.7. The user shall be able to stop the program at any time during the test.

4.8. After the program times out or the user stops the program, it will display accurate data transfer speed in Kbps/Mbps/Gbps.

5. RESOURCES

5.1. Non-Functional Requirements

5.1.1. The Java programming language will be used to create the program. This language was chosen because each member has experience working in it and it has all the tools we need to create a quality program.

5.1.2. Java Development Kit 8

5.1.3. JFrame for the GUI

5.1.4. The program will be multi-platform (Works on Windows, Mac and Linux)

5.1.5. Github will be used to collaborate between each other: https://github.com/aaron-monahan/Network-Speed-Tester-