Project Courier

Formerly Project Awesome

Stefan Couturier
Abhinav Gambhir
Andrew Stansel
Matt Stone

December 12, 2015

1) Introduction

1.1 Context

File sharing has become an important aspect of our day to day internet use, so we wanted to create an application with the basic functionality of sharing files over the internet. Project Courier allows multiple users the ability to download files easily from both a central server, and directly from other users.

The application uses the Socket Application Programming Interface (Socket API) and the Transmission Control Protocol (TCP) to allow different hosts the ability to communicate over an Internet Protocol (IP) network. The Socket API is provided by the Operating System (OS) to allow the application the use of network sockets, and a TCP connection is used to provide a reliable, order-dependent delivery of data streams between hosts. Program Courier also uses Peer-to-Peer (P2P) file transfer, which is setup automatically by the server upon request from a user.

1.2 Problem Statement

Earlier in the course, for the assignment, our team built a simple file sharing system that was to be used by one server and one client. Our goal for this project was to improve upon that system by adding more functionality, such as multiple users, an easy-to-use Graphical User Interface (GUI), and the ability to chat with other users. The idea was that multiple peers could upload particular files that they thought others might enjoy to a central server. However, the application would also have a chat board so that users could talk to each other, and also request particular files that they might want from other users. This would allow users the option of uploading a requested file to the central server, or notifying other users that they had a particular file so that others could download directly from them. In this way, users would effectively be using the server as a way to download files, search for files, and establish P2P connections for file transfer.

1.3 Result

The highlight of our project is the server connecting multiple clients together, acting as a sort of concierge, by providing clients many files that they can download, and also by giving a client the ability to download directly from another client. The program successfully implements Threads allowing for multiple clients to connect to the server. We also succeeded in having the clients able to initiate connections in between themselves allowing Peer-to-Peer (P2P) file sharing. This happens by the clients using the server to talk with each other and opening up a new socket that links the two clients together. When they are finished the socket terminates and closes. Our application also allows clients to chat to one another, which adds an additional search functionality to the application.

1.4 Outline

In reference to our problems, the following outlines our process of working towards our goals and how our project came to completion.

a) Section 2 contains background information on the concepts that we have used.

b) Section 3 describes in detail how said concepts were used to build the project.

c) Section 4 explains how we were able to evaluate the quality of our project.

d) Section 5 summarizes the process and results of building our project.

2) Background Information

Project Courier is an extension to the assignment that we did earlier in this course. The premise is the same, but we improved upon its functionality by connecting multiple users to the server and allowing them to share files amongst each other. Since there is one central server and multiple clients connecting to it, threads were introduced to the server to allow it to manage all of the clients. Also, the whole system has been incorporated with a GUI which allows for smooth use by anyone.

While peer-to-peer file sharing is nothing new, we thought our application was somewhat unique in how you have the ability to post requests for files directly inside the application. This allowed clients to use the server as a way of searching for files amongst peers who likely have common interests, in addition to just searching the central repository of files found on the server. Regarding our P2P file transfer, we really did not do any research into how other applications handle that functionality, as we wanted to keep our application simple, but also write it completely ourselves. So for our P2P functionality, we came up with our own way of implementing it, which may or may not be similar to how most other applications handle it (a good description of our implementation is found in Section 3.3, which also includes a sequence diagram of the process). Similarly, we came up with our own way of handling the communication between the clients and the server. It is a simple process (described in Section 3.2) where we send unique string identifiers back and forth between the clients and server to help control the flow of communication between the two, and get each side ready for what type of data it will need to send or receive.

3) Result

The application that we created was easy to use, and had all of the functionality that we intended (as explained in Section 1.2). While using the application the user interacts with 2 different interfaces. Most of the time spent by the user is with the main application interface, as this is where all of the file transferring and chat functionality takes place. However, to begin the application, the user must first enter some information.

3.1 User identification

When logging onto the server, every user has to provide a username for identification purposes. The first window that opens asks for three details; the IP address of the server, the username, and the directory which they wish to share from. After this information is processed, the new user is connected to the server and its information is relayed to all the other users that are online so they can browse their directory.

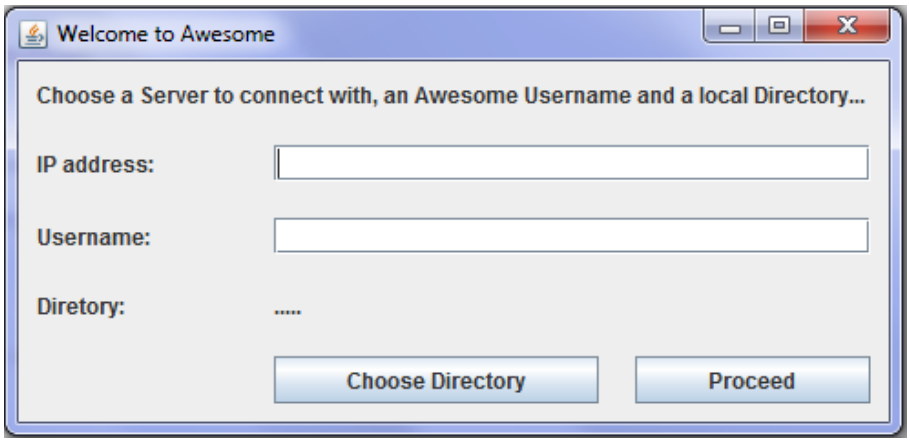

Figure 1: The login screen, where the user must choose an IP address to connect to, a username, and a local directory to exchange files from

3.2 Multiple connections

As mentioned earlier, we used threads in order to manage multiple connections to the server. Each connection has a minimum of two dedicated threads, a server thread which is processing all the information and handling the connection, and a client thread (representing a user) whose job is to listen for commands from the GUI and send appropriate requests to its server thread.

When a user makes a request by entering some input through the GUI, that input is detected by the client code, and if all necessary conditions for that request are met locally, the client sends a corresponding notification to the server thread to initiate a transfer of some kind. If the particular request is to download a file from the server, the client code first verifies that the requested file does not already exist locally. If it does not, the client sends a simple, but unique, text identifier to the server thread to allow it to properly prepare for what it needs to do to satisfy the request. This unique identifier is used to help facilitate the proper flow and communication between the client and server. In this example, the client would send the string identifier “download”, which would allow the server thread to get ready to receive the filename of the file to be sent to the user. The server thread would then notify the main server to get the proper file, the server thread would send the unique identifier “download” back to the client thread to notify it to get ready to receive a file, and then the file transfer would begin.

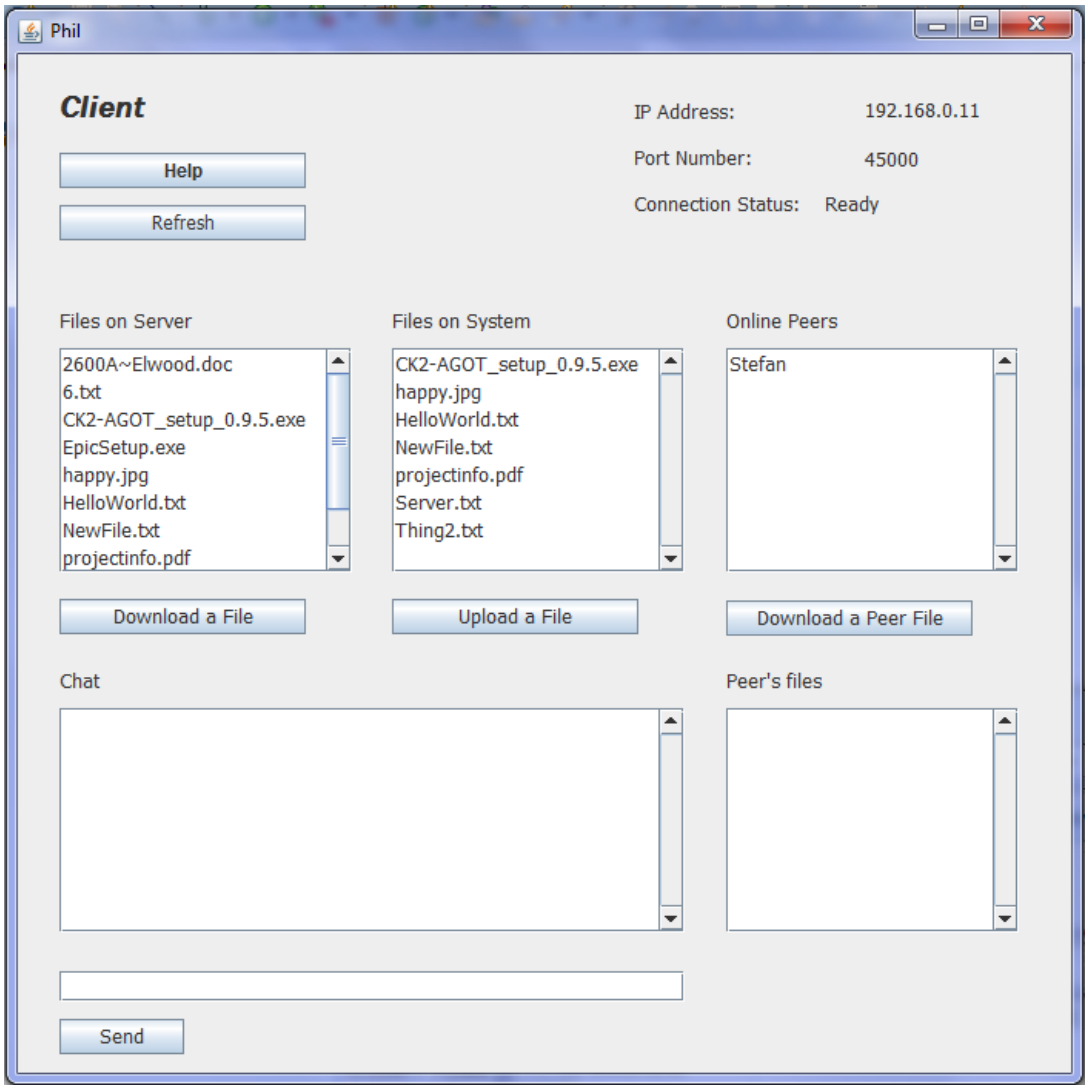

Figure 2: A client has logged in and can see another peer connected to the server

3.3 Peer-to-Peer

To make file sharing easier and faster, we added the functionality of peer-to-peer connections which would allow two users to share a file between each other without having to go through the server. These connections are only temporary and are terminated as soon as the file is shared. When Client 1 (the client requesting a file from a peer) clicks on the name of Client 2 (an online peer), the files of Client 2 get displayed in the 'Peer's Files' list of the GUI.

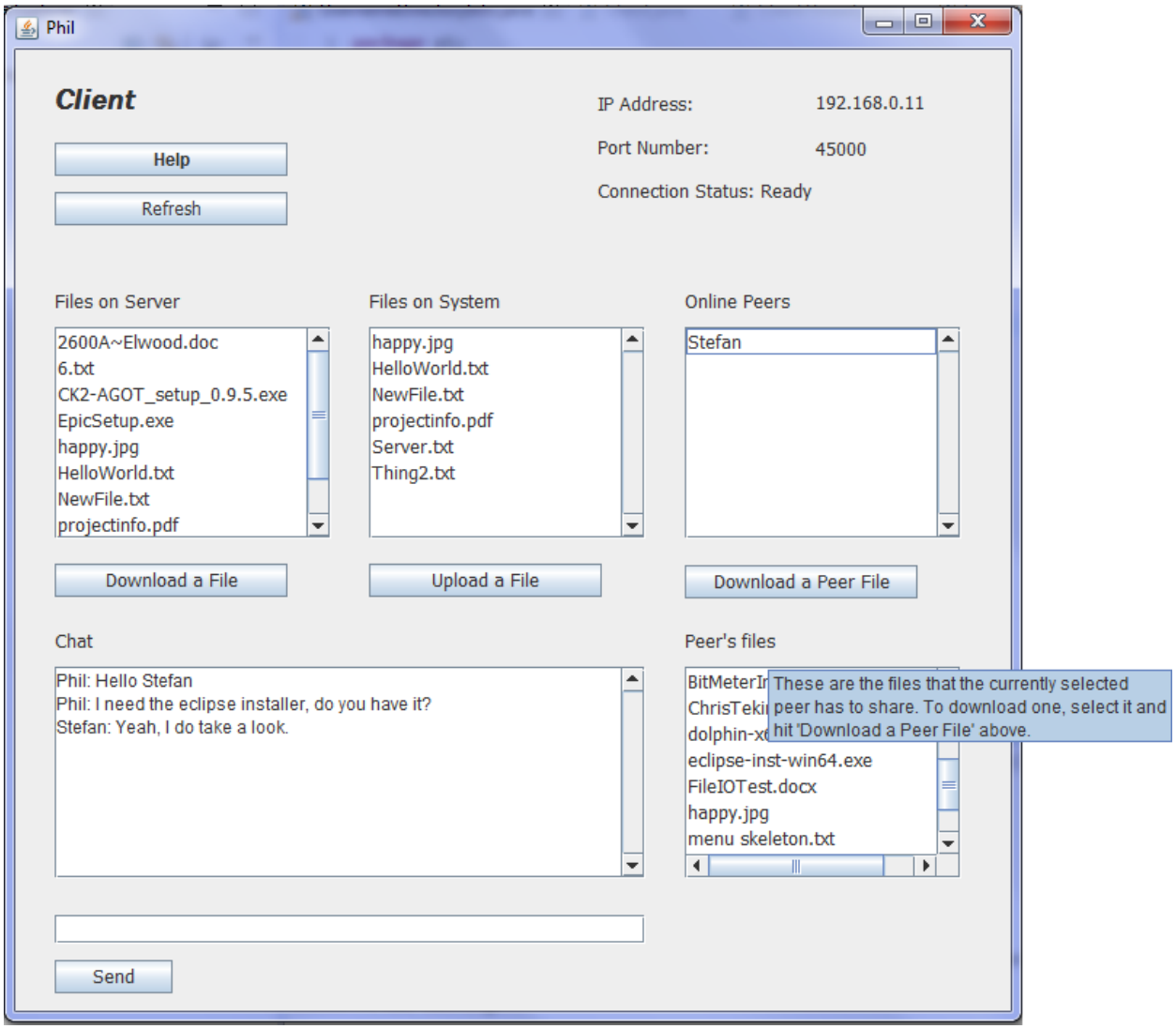

Figure 3: By selecting a peer, their files appear in the 'Peer's files' list

Client 1 can then download one of those files by selecting the file and pressing the 'Download a Peer File' button (as shown in Figure 3 above, and explained in section 3.5, hovering the mouse over certain areas of the GUI will display a 'help' message). The sequence of steps involved in this process are shown in the sequence diagram below.

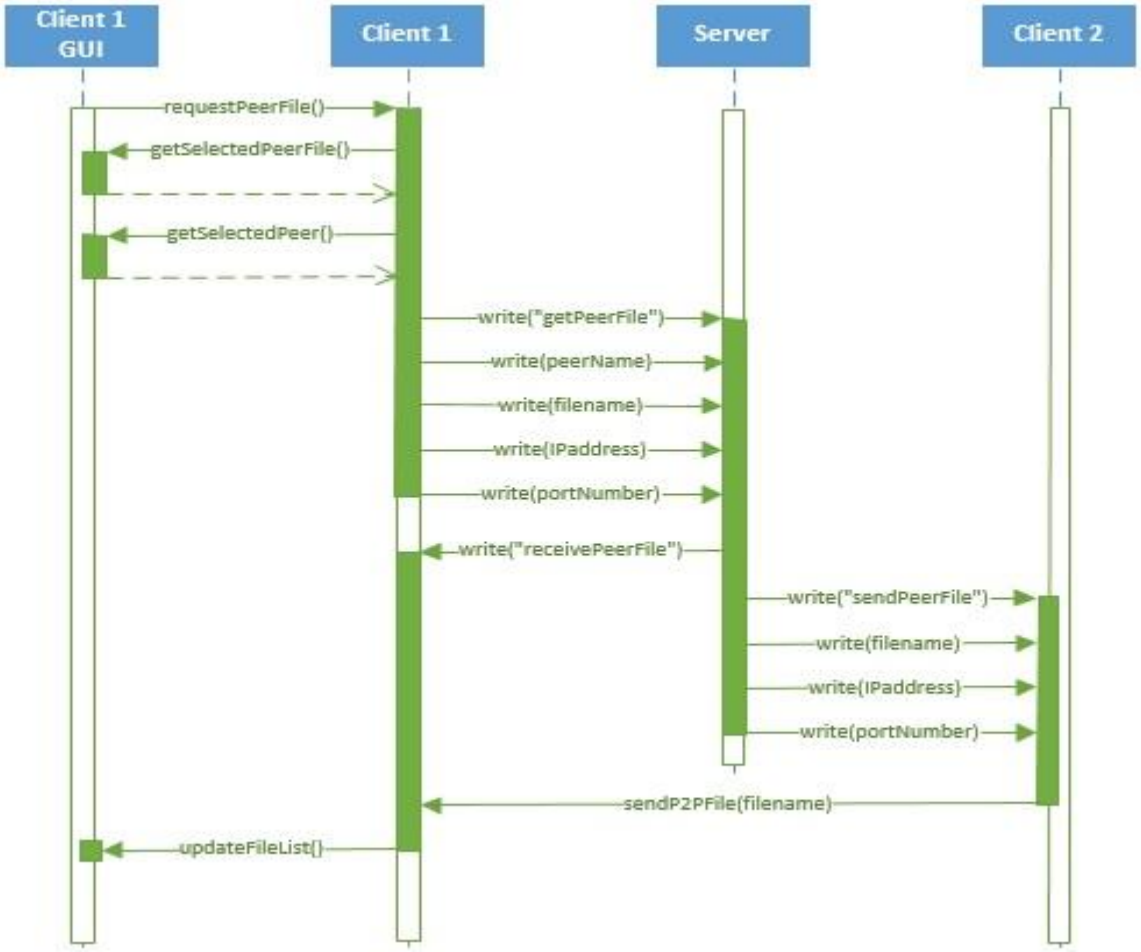

Figure 4: The sequence of steps involved in a peer-to-peer (P2P) file transfer

Once the GUI notifies the client thread that Client 1 has requested to download from Client 2, the client code gets the currently selected peer file and the currently selected peer (Client 2) and then sends the unique identifier "getPeerFile" to the server thread. The server thread is then ready to receive the information that it will need to relay to Client 2 to allow the direct peer-to-peer file transfer. Client 1, who is requesting the file, will send the username of Client 2 (so that the server thread can notify the server of who it needs to send the information to), the filename of the requested file, and its own IP address and available port number. With this information, the server has everything it needs to send to Client 2 (who has the file being requested by the initial client), and it knows who Client 2 actually is.

The application handles P2P file transfer by setting up completely new connections on different ports, separate from the threads used for the rest of the application. In this example, Client 1 (who is the recipient of the file) sets up as a temporary server, and Client 2 (who will be sending the file) sets up as a temporary client. The server begins this process by sending a message to Client 1 to let it know that it should setup to receive a file from a peer. It then opens a connection on a particular port (which it has provided to Client 2) and waits for a connection to

be established from Client 2. Once the server has notified Client 1 to prepare to receive a file, it then sends Client 2 the filename of the requested file, and the IP address of Client 1, along with the port number that Client 1 is listening on. Client 2 is then able to directly establish a connection with Client 1 and send the file. As soon as the file is transferred, each side closes this temporary connection, and resumes functioning on its main threads.

3.4 Chat

This functionality allows the clients to talk to each other when they are connected to the server. The process for this was fairly simple; a client types a message and hits send, the message is the sent to the server which in turn relays the message to every client and ends up on the chat window for everyone to see. Clients can simply use the chat functionality to have conversations amongst any online peers, or they can use it to request files. Since users are bound to have many files on their computer other than the ones present in their current directory, making a request within the chat window is a simple way of requesting that a peer either upload a file to the server, or just simply add it to their directory to allow it to be downloaded.

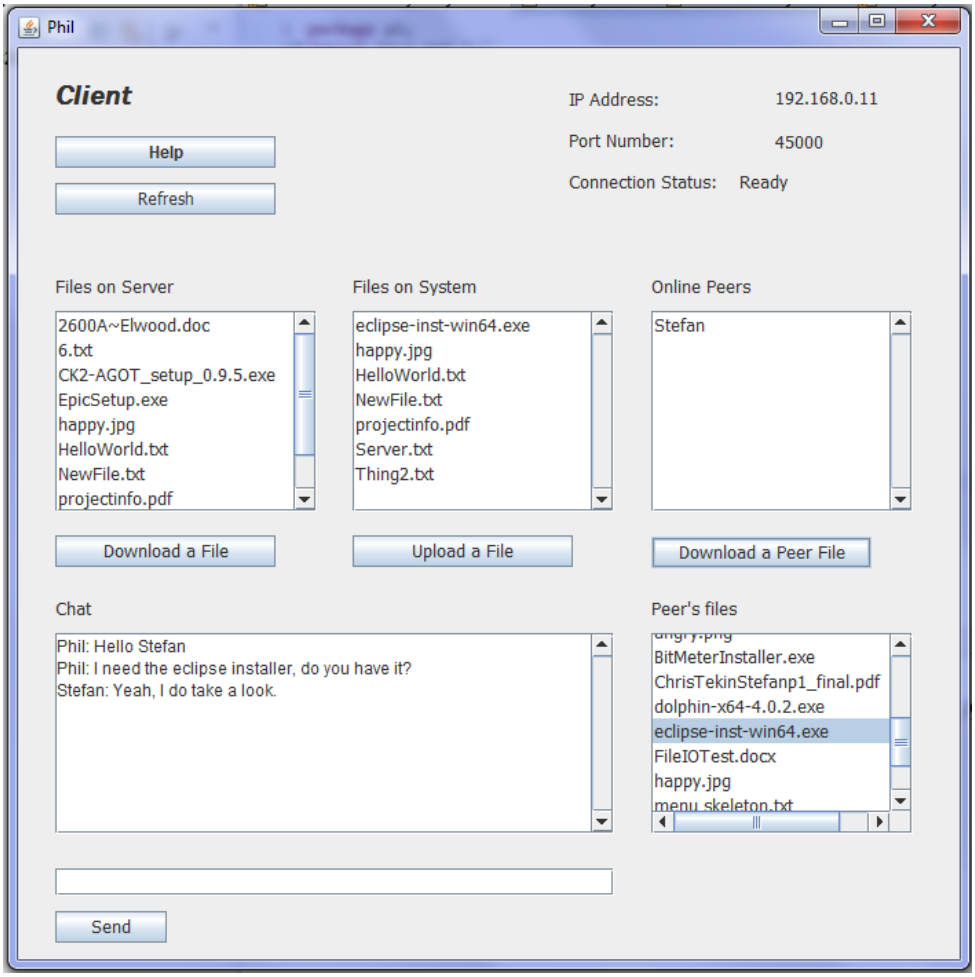

Figure 5: Within the Chat window, clients can both talk and request files that they wish to download

3.5 Help for Users

Since we wanted the application to be easy to use, we thought adding some descriptions of the functionality to the interface would be useful. Of course, we did not want to clutter the interface with lots of text, as we wanted a nice, clean look. We also figured that the learning curve to using the application would be very small, and if users did not immediately understand how to use it, with a little help, they would quickly find it very easy to use. To help new users learn how to use the application, we added hover text to the buttons and a couple labels (an example is seen in Figure 3 above) which describe the functionality of pressing buttons, and explain what is being displayed in certain areas. To help make sure that users realized this, we added a 'Help' button which displays a message to notify users of this functionality.

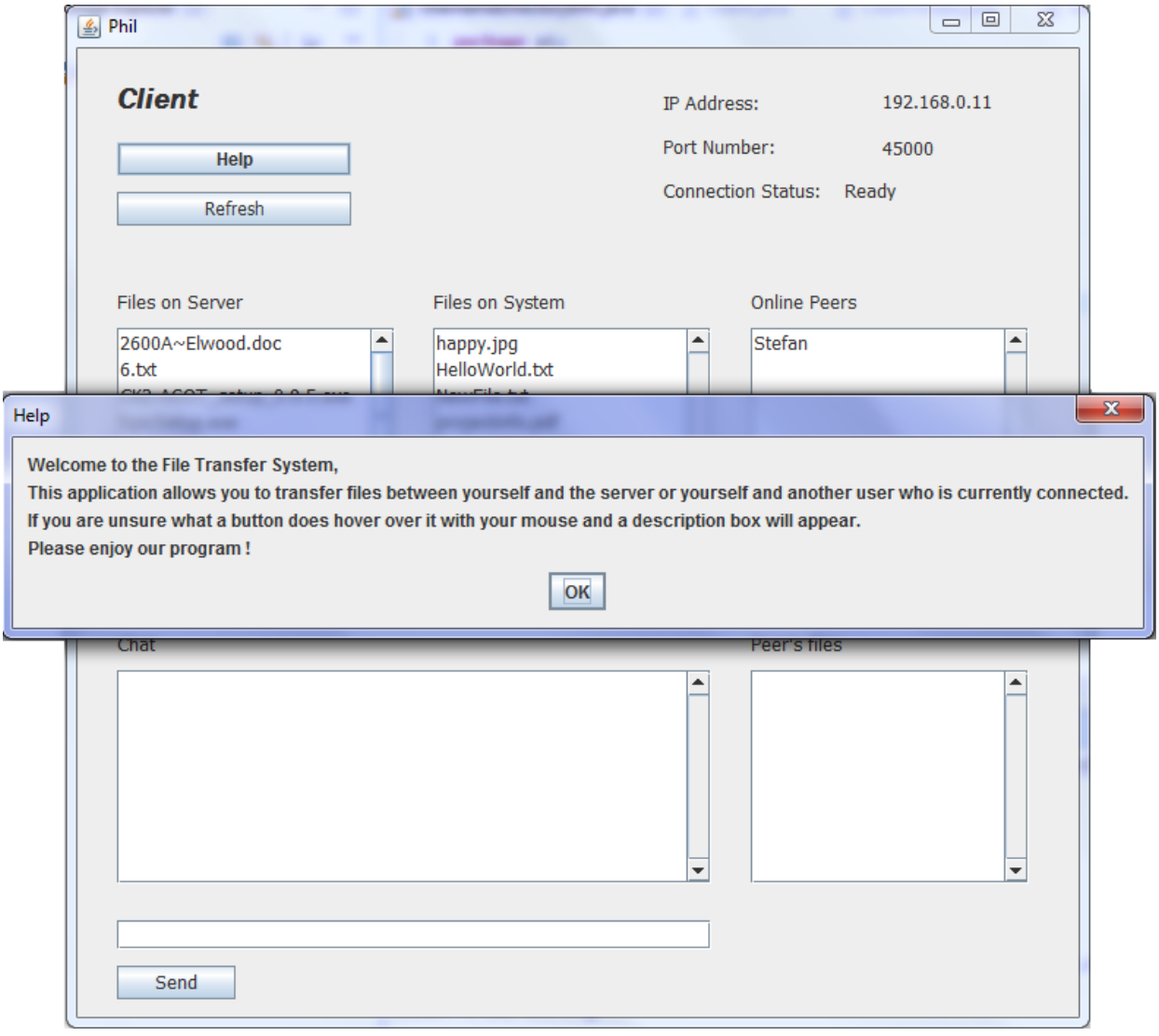

Figure 6: If the user clicks the 'Help' button, this message appears in order to help new users understand how to use the application.

3.6 Server Interruptions

Since it is possible for the server to be terminated without the clients knowing it is necessary to handle this event. In the case of the server being terminated, it sends out a signal that tells every single client currently connected that the server is closing. This then triggers the clients themselves to shut down after alerting the user.

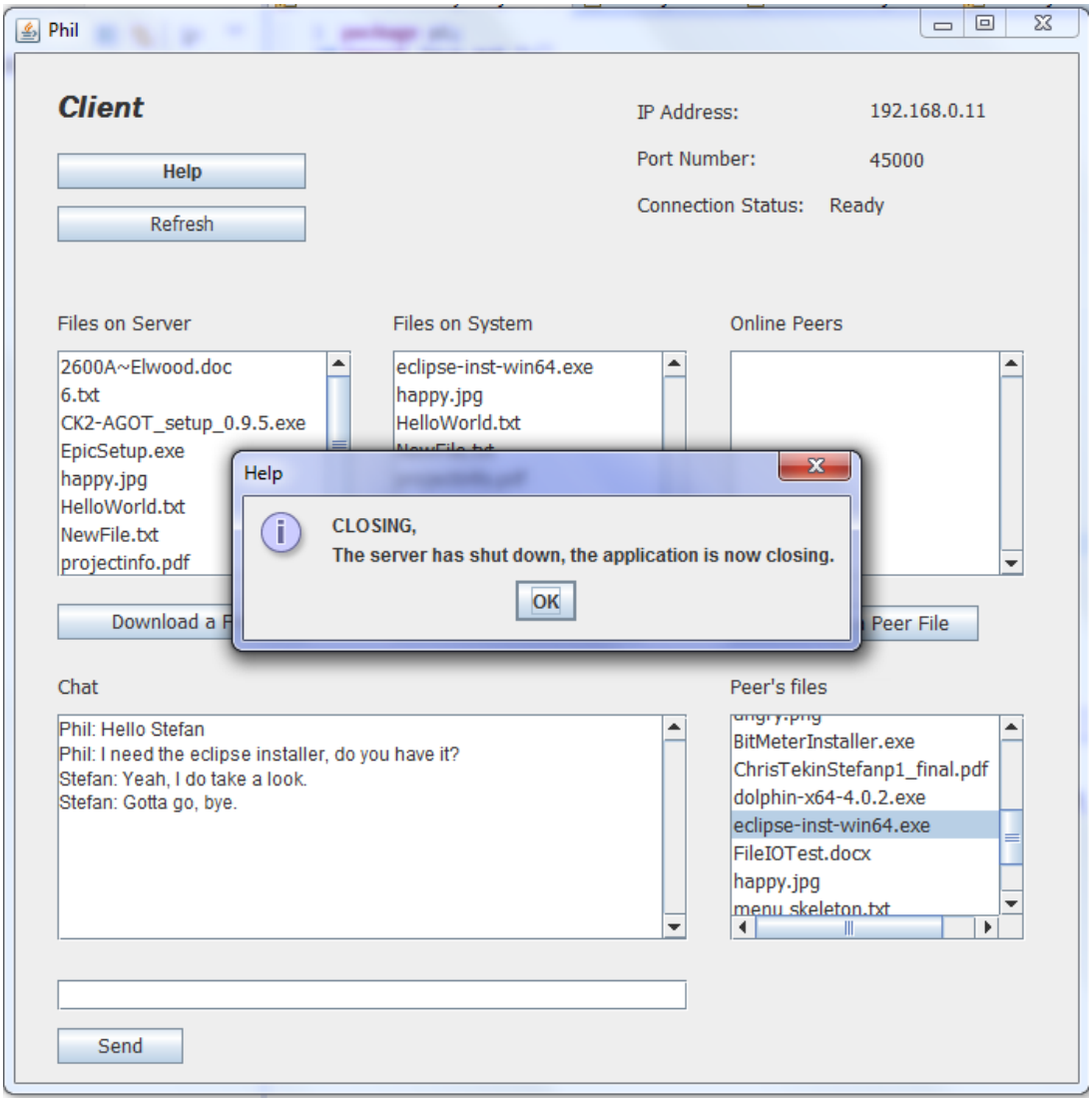

Figure 7: After receiving a TERMINATE signal from the server, a dialog box notifies the user the application is closing.

4) Evaluation

The purpose of our application was for it to give all of the functionality that we wanted while being very easy to use, and nice and simple to look at. Besides our group putting in the time and attention to detail to try and achieve this, we also created both a Pre-Test Questionnaire and a Post-Test Questionnaire (found in Appendix A and Appendix B respectively, at the end of the report). The purpose of these questionnaires was to gain a little insight into people's tendencies and usage regarding how they exchange files over the internet, and also to rate the quality of our applications functionality and ease-of-use. Each member of our group was able to get a couple family members to test the application, and we were also able to get several peers from school to test the application to help provide a more balanced sample space.

The Pre-Test Questionnaire was given to participants just before testing our application. In general, everybody tended to download files from the internet using 3 to 4 different applications (including downloading torrent files for various media, exchanging files through email, and downloading music). Most participants indicated that the current applications they use had an interface that they liked, but that they would enjoy being able to accomplish everything through one application, and that they would enjoy being able to make simple request to other peers of theirs for files that they might enjoy.

The Post-Test Questionnaire was given to participants just after they had used our application. The portion of our testers that were family members, who didn't have much in-depth knowledge of computers, did not know how to find their own IP address, and so they weren't entirely comfortable with having to enter the server's IP address to be able to connect with it. Users of the application don't ever need to be able to find out their own IP address, as our P2P functionality is all done automatically by the application, however, we thought it an interesting question to ask since they at least need to enter the IP of the user acting as the server at the login screen. During our initial testing with family members, we did not have any helpful hover messages explaining the functionality of our application, and most users found it a little bit difficult to understand how to upload and download files, especially from the 'Peer's file' list. In response to this, we added in a 'Help' button and descriptive hover messages that appear when the mouse hovers over certain parts of the GUI. After making this change, we tested with the same family members, and also several peers from school, and also gave them our Questionnaires to fill in. Regarding the question asking the participants if they understood the purpose of picking a local directory, 100% of participants indicated that they did understand because of our hover messages which get displayed by holding the mouse over specific areas of the GUI. Similarly, 100% of users were quick to notice the 'Help' button in the main application which helped them make use of our hover messages and understand the user interface and how to use the application properly. About half of the users noticed the application seem a bit unresponsive once or twice, and this is because each client only uses one thread, and so when performing one task, the GUI is temporarily disabled until the operation is completed (as noted in section 5, this is an area for improvement in possible future work on the application). About

80% of users rated the application a 4 out of 5 in overall quality and indicated that they would enjoy using the application again.

5) Conclusion

5.1 Project Summary

The objectives that we set out to achieve with this project were all met. Our objective was to create a simple application so that users could essentially 'meet' online and exchange files amongst themselves. We intended to have one server act as a central file repository, and then allow the server to give clients the information they needed to connect directly to each other to allow P2P file transfer. The intention of this was so that not all files needed to be on the server, and if one user had yet to upload a file to the server, another client could download a file from them directly, to make the process easier and faster (since only one transfer would need to happen instead of two). We were able to get this accomplished, and the result seemed to be that it made the overall process of getting various types of files very easy. We also provided a chat feature, which allowed users to post comments to a global chat board. This feature was also intended to allow users to make requests for files that they want but don't see on the server or on any file lists of online peers.

5.2 Areas for Improvement

We did manage to accomplish our goals with the application, however, there are certainly areas for improvement that could be addressed in future work. One such area would be making the application be able to use multiple threads. Currently, when a user makes any request through the GUI, it becomes temporarily disabled while the requested process completes. If a user is downloading a large file, they need to wait for that file to download completely before they can use the GUI again. It would be a nice addition to be able to search for more files or post to the chat board while a download is in progress.

Another area for improvement would be having download and upload progress bars, with the ability to pause downloads and resume them again later. For downloading most files, it may not really be an issue, but especially for large files where a peer is potentially far away, a file transfer may be quite slow. If for some reason a user needs to shut down their application before a download finishes, it would be great if they had the option of pausing a download and resuming it again when convenient.

Contribution of Team Members

Stefan Couturier: Worked on the CHAT function of the program, also made it so when there were changes detected by the server it would update all clients and edited the GUI a little. Documented the program flow and interface of our application for the Project Report.
Abhinav Gambhir: Designed the GUI for the client, and helped write and setup the foundation and structure of the Project Report.
Andrew Stansel: Created the login GUI, implemented P2P file transfer and added description messages that appear when hovering over a button. Helped setup the control of communication between the client and server threads, and communication between the GUI and the client.
Matt Stone: Worked on the actual transfer of data between the server and clients. Made the application implement Threads to accommodate multiple Clients, and also worked on P2P file transfer. Helped setup the control of communication between the client and server threads.

Appendix A

Pre-Test Questionnaire:

1) Do you ever download files from the internet, such as torrent files of movies or music?

YES NO

2) Do you think that friends sharing files over the internet is bad or socially unacceptable?

YES NO

3) If you use an application for downloading or sharing files over the internet, do you enjoy its interface and ease of use?

YES NO N/A

4) How many different applications do you use to download, send, or transfer files of any kind over the internet?

1 2 3 4 5 +

5) If you answered any number greater than '1' to the previous question, would you enjoy having one simple application for your computer that allowed you to send any kind of files to friends or family?

YES NO N/A

6) Would you enjoy the ability to be able to make simple posts to an online forum requesting particular files (movies, for example) that you might enjoy having?

YES NO

Appendix B

Post-Test Questionnaire

1) Would you know how to easily find out your own IP address?

YES NO

2) When at the login window, did you understand the purpose of picking a local directory before entering the application?

YES NO

3) Did you think that the User Interface of the application was informative?

YES NO

4) Did you notice the 'Help' button, and if so, did you click it?

YES NO DIDN'T NOTICE IT DIDN'T NEED IT

5) Was it easy to understand, or figure out, how to use the Application?

YES NO

6) Did the application ever seem unresponsive?

YES NO

7) Were you able to upload to, and download from, the Server?

YES NO

8) Were you able to download a file directly from a Peer?

YES NO WAS THAT POSSIBLE?

9) Overall, how would you rate the quality of the Application? (5 being best)

1 2 3 4 5

10) Would you want to use the Application again?

YES NO MAYBE