Project 1

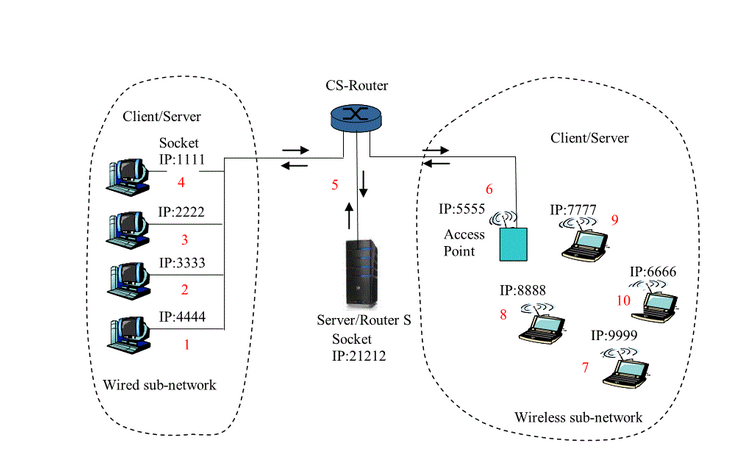
Robert Morris

Shakkeim Saunders

Liudas Tumasonis

Ramkumar Vedagiri

Clay Casto



**Introduction:**

**Client** would initiate requests to servers and would wait and receive replies. The client would usually connects to multiple servers at one time and would interact with end users using user-interfaces like a GUI.

**Server** would usually wait to get a request from client and would reply with the requested data or information. servers would communicate with other servers to get the requested data. routers would connect multiple networks to communicate with one another

**Sockets:**

A **network socket** is an endpoint of an inter-process communication flow accorss a computer network.

A **socket API** is an application programming interface usually provided by the OS, that allows application programs to control and use network sockets.

A **socket address** is the combination of an IP address and a port number.

**IP Address**  is a numerical label assigned to each device (e.g., computer, printer) participating in a computer network that uses the internet protocol for communication

**port number**: A port is identified for each address and protocol by a 16-bit number

**Routing table** to find the destination would contain information such as IP address and port number.

**Flow for project**:

For this project, the client would send a request to the Client/Server router, which would then send the request to the Server/Router. Once in Server/Router the Server/Router routing table looks for the destination of the message it will send the message to the other Client/Server router. The response would go back in the reverse route.

**Client**

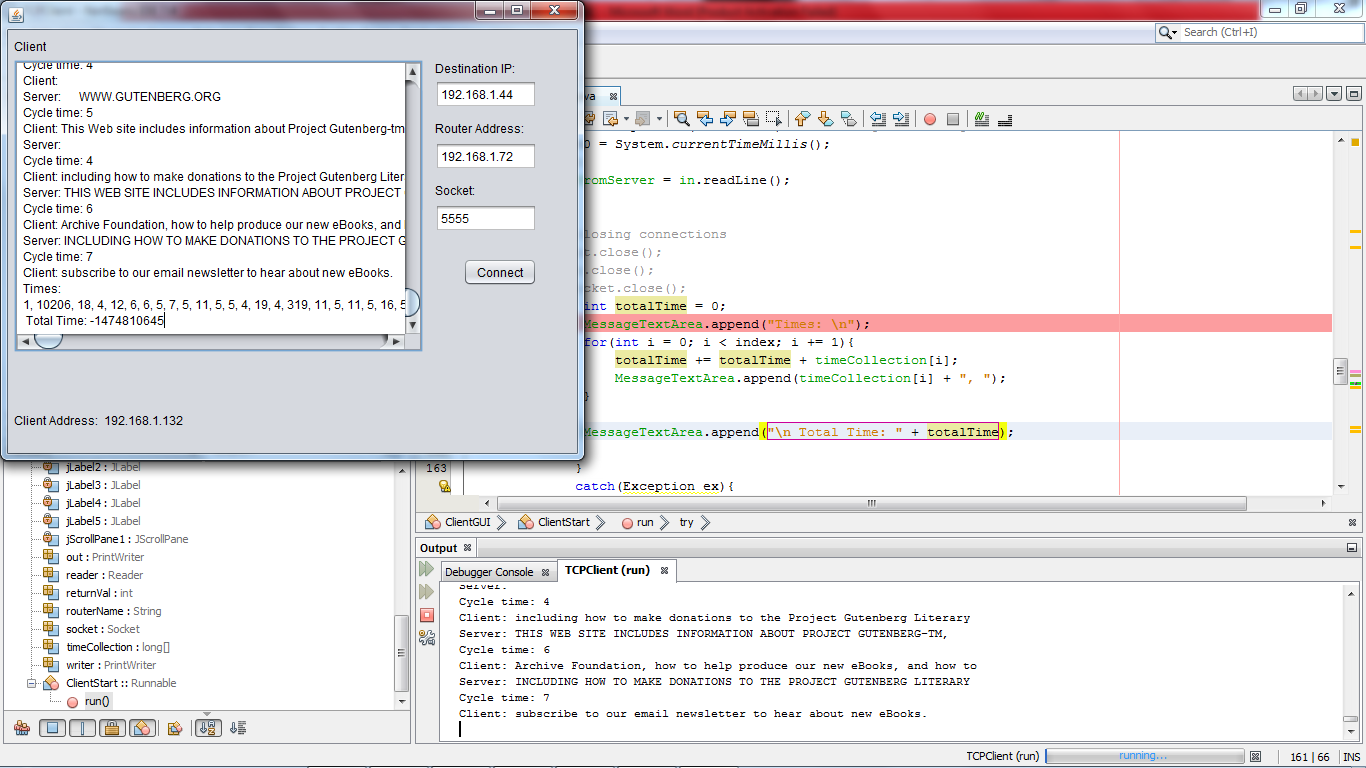
The client model was very easy to implement. It started with a basic TCP client setup and no GUI that acted as a chat client then evolved. It has your basic components:

* Stream Socket for communication. Uses all default properties that comes with a stream socket. It is a client socket. Points towards the InetAddress of the server router and the specified port.
* PrintWriter for writing messages into the socket that will then be sent to the server router.
* BufferReader for reading incoming messages from the server router.

These are the three main components. The other components are supporting components to these three. For phase one of the project we designed the client to read a file then send it line by line to the designated receiver. The receiver takes what is sent, capitalizes it, and then sends it back.

The way the client works is by first connecting to the server router. We use the stream socket to communicate with the server router. The server router IP and Port number are specified in the GUI. The receiver of the file is distinguished by the forwarding IP that is inserted via the GUI. When you click connect the application begins the connection process. It takes in the information passed to it via the GUI and sets IP to connect to and port to use to communicate with the server router. We then set out PrintWriter equal to the socket output stream so we can send messages. Then set the BufferedReader equal to the input stream for incoming messages. If connection is successful, an open file dialog box will appear. Here you can select the file you want to send. As of right now the code is only able to handle plain text. We set a FileReader to read in the file, and then pass it to a BufferedReader so we can read it out line by line to our output stream. The initial send is the address the client is going to pass the file too. The client then waits for a response from the server router verifying we have a good connection. Then the client sends his IP for the server to save for a return address. This is the data that the server router will use in its routing table. We now get into the message passing of the file itself. A timer is started to record cycle times from being sent to server and then back to the client capitalized. This is all outputted to the message text box we created along with the cycle time. In a better break down, we read a line from the file. Then using our print writer outputs the line to the stream. Then the client waits for the capitalized message to come back. This continues until the end of the file. Using a array, all the cycle times are collected. Once the entire file has been transferred and received we output a list of all the times and calculate the total time in milliseconds.

Client Run with a 87.6kb plaintext file:



Connecting to Router: 192.168.1.72 Socket: 5555

Connection successful

ServerRouter: Connected to the router.

Server: Connected to the router.

Cycle time: 1

Client: The Project Gutenberg EBook of The Pioneer Trail, by Alfred Lambourne

Server: NULL no response yet.

Cycle time: 10206 initial send. Delay built in router created longer time.

Client:

Server: THE PROJECT GUTENBERG EBOOK OF THE PIONEER TRAIL, BY ALFRED LAMBOURNE

Cycle time: 18

Times in milliseconds:

1, 10206, 18, 4, 12, 6, 6, 5, 7, 5, 11, 5, 5, 4, 19, 4, 319, 11, 5, 11, 5, 16, 5, 8, 6, 10, 6, 4, 8, 7, 6, 5, 6, 5, 4, 5, 5, 5, 6, 5, 4, 9, 13, 9, 7, 5, 5, 9, 5, 4, 4, 7, 5, 6, 6, 4, 5, 5, 6, 5, 10, 7, 5, 6, 4, 6, 5, 5, 6, 5, 5, 6, 6, 4, 6, 3, 5, 5, 5, 7, 5, 5, 17, 7, 5, 8, 4, 6, 6, 5, 5, 5, 5, 4, 5, 10, 11, 4, 6, 4, 6, 8, 5, 4, 5, 4, 5, 6, 5, 6, 4, 10, 4, 6, 5, 8, 5, 3, 6, 5, 5, 6, 5, 5, 8, 4, 5, 14, 5, 5, 5, 6, 4, 5, 6, 7, 4, 5, 5, 3, 6, 5, 5, 3, 4, 6, 5, 4, 6, 3, 7, 4, 5, 13, 8, 5, 5, 7, 4, 4, 5, 5, 7, 5, 5, 4, 7, 4, 5, 6, 6, 4, 5, 4, 4, 8, 5, 5, 5, 4, 4, 6, 5, 4, 4, 4, 5, 3, 4, 6, 15, 3, 5, 4, 7, 4, 4, 4, 4, 9, 6, 4, 6, 6, 10, 5, 5, 4, 5, 4, 6, 4, 7, 4, 5, 4, 4, 5, 6, 4, 6, 4, 5, 5, 5, 7, 4, 4, 4, 4, 5, 6, 4, 6, 6, 10, 6, 3, 5, 7, 5, 4, 5, 4, 5, 3, 6, 4, 5, 7, 4, 5, 5, 3, 6, 5, 4, 4, 9, 4, 10, 4, 5, 4, 4, 4, 4, 9, 4, 5, 5, 5, 7, 6, 5, 5, 4, 5, 7, 9, 5, 6, 4, 6, 4, 5, 4, 5, 5, 4, 8, 5, 4, 3, 5, 6, 4, 8, 4, 3, 5, 5, 5, 5, 7, 5, 5, 4, 4, 8, 5, 6, 5, 7, 4, 4, 7, 6, 5, 5, 6, 6, 7, 4, 12, 5, 6, 6, 4, 4, 7, 4, 4, 8, 4, 17, 5, 6, 5, 6, 4, 11, 6, 5, 4, 9, 3, 5, 7, 5, 6, 4, 4, 6, 7, 5, 4, 5, 5, 23, 17, 7, 7, 9, 11, 6, 21, 5, 5, 6, 6, 10, 5, 6, 4, 5, 5, 5, 7, 4, 4, 4, 4, 6, 6, 4, 5, 5, 5, 6, 4, 5, 6, 6, 4, 5, 4, 5, 8, 5, 4, 5, 4, 7, 5, 4, 5, 4, 7, 4, 3, 6, 4, 6, 5, 4, 4, 5, 7, 5, 4, 6, 4, 8, 5, 5, 4, 5, 6, 8, 20, 6, 4, 4, 8, 4, 4, 4, 11, 13, 4, 6, 4, 5, 6, 4, 5, 6, 5, 5, 8, 4, 7, 5, 6, 6, 5, 4, 5, 6, 6, 5, 5, 5, 3, 6, 4, 6, 7, 4, 5, 4, 4, 7, 12, 4, 4, 7, 5, 7, 4, 134, 4, 4, 4, 4, 6, 5, 4, 5, 6, 6, 4, 5, 11, 4, 5, 4, 6, 4, 5, 235, 323, 49, 4, 7, 5, 6, 6, 7, 4, 6, 6, 230, 327, 18, 5, 6, 6, 5, 4, 7, 7, 4, 5, 7, 8, 3, 4, 4, 117, 5, 5, 4, 12, 5, 9, 12, 10, 6, 5, 7, 6, 6, 7, 230, 336, 7, 12, 5, 4, 6, 5, 5, 11, 5, 4, 8, 5, 4, 4, 231, 4, 11, 6, 7, 5, 5, 5, 6, 4, 4, 4, 3, 7, 5, 4, 7, 231, 3, 213, 4, 8, 21, 5, 7, 5, 6, 8, 6, 4, 7, 4, 7, 4, 235, 6, 16, 11, 7, 14, 3, 14, 5, 4, 4, 233, 32, 6, 18, 4, 4, 5, 5, 4, 4, 4, 6, 4, 4, 7, 4, 4, 3, 161, 4, 7, 10, 6, 6, 5, 4, 6, 8, 4, 5, 5, 4, 4, 4, 6, 258, 178, 7, 4, 6, 4, 9, 6, 4, 5, 5, 4, 3, 5, 4, 6, 4, 6, 6, 5, 4, 6, 5, 5, 4, 4, 5, 5, 4, 3, 5, 4, 4, 6, 4, 4, 5, 4, 5, 6, 4, 5, 7, 5, 5, 4, 6, 4, 7, 4, 4, 4, 6, 5, 6, 5, 5, 4, 4, 6, 5, 3, 7, 7, 5, 4, 4, 5, 5, 4, 5, 6, 6, 6, 6, 4, 5, 3, 6, 4, 3, 7, 4, 6, 4, 4, 5, 6, 4, 4, 4, 4, 4, 5, 4, 4, 4, 5, 5, 6, 4, 5, 6, 6, 10, 4, 5, 4, 7, 5, 5, 6, 3, 4, 7, 5, 15, 5, 7, 4, 10, 6, 5, 4, 6, 4, 5, 4, 5, 5, 5, 4, 6, 5, 5, 4, 3, 7, 4, 7, 4, 8, 4, 7, 15, 12, 25, 8, 4, 5, 5, 6, 6, 9, 5, 8, 6, 4, 6, 5, 4, 5, 6, 4, 5, 6, 6, 6, 5, 9, 13, 7, 6, 23, 6, 4, 3, 6, 5, 5, 5, 4, 6, 3, 7, 10, 6, 5, 6, 7, 6, 4, 23, 4, 4, 4, 11, 4, 4, 4, 4, 5, 6, 6, 14, 6, 5, 5, 6, 4, 5, 5, 4, 10, 6, 4, 5, 6, 3, 11, 5, 5, 8, 4, 4, 8, 7, 4, 5, 4, 4, 5, 9, 3, 4, 5, 4, 4, 4, 4, 6, 6, 4, 4, 7, 5, 4, 9, 4, 13, 9, 6, 5, 5, 4, 10, 7, 4, 5, 9, 5, 5, 3, 5, 4, 4, 4, 4, 5, 4, 5, 5, 5, 4, 5, 7, 4, 5, 6, 4, 6, 5, 4, 5, 5, 4, 4, 4, 4, 4, 6, 4, 4, 4, 8, 7, 5, 4, 7, 5, 4, 4, 3, 5, 5, 10, 4, 4, 5, 5, 6, 4, 6, 5, 7, 4, 4, 6, 4, 4, 4, 5, 5, 5, 4, 4, 4, 4, 4, 9, 5, 4, 4, 5, 13, 4, 5, 7, 4, 4, 4, 5, 4, 5, 7, 4, 5, 4, 4, 5, 4, 5, 5, 8, 4, 14, 6, 5, 4, 4, 5, 13, 4, 4, 6, 3, 5, 4, 4, 4, 8, 4, 5, 6, 5, 9, 9, 4, 6, 3, 4, 4, 3, 5, 4, 4, 4, 5, 9, 4, 4, 6, 8, 3, 5, 4, 4, 5, 8, 4, 4, 4, 11, 6, 4, 4, 5, 15, 6, 5, 5, 4, 5, 4, 5, 4, 4, 4, 4, 6, 5, 6, 5, 5, 6, 7, 3, 4, 7, 5, 4, 6, 4, 4, 5, 7, 4, 5, 4, 5, 6, 4, 6, 5, 4, 5, 6, 5, 8, 5, 6, 4, 5, 4, 5, 6, 5, 5, 6, 9, 6, 4, 4, 5, 7, 4, 4, 9, 4, 5, 5, 7, 6, 4, 6, 4, 5, 19, 5, 5, 7, 4, 5, 9, 6, 4, 5, 4, 8, 6, 4, 6, 5, 7, 6, 5, 4, 5, 6, 5, 5, 9, 4, 4, 6, 4, 6, 6, 5, 5, 5, 6, 5, 5, 6, 8, 6, 3, 5, 5, 11, 4, 13, 6, 3, 4, 4, 5, 4, 5, 5, 4, 3, 4, 7, 4, 5, 7, 5, 4, 9, 4, 5, 4, 5, 4, 5, 4, 4, 4, 5, 4, 4, 5, 6, 4, 4, 6, 5, 6, 4, 4, 5, 5, 5, 4, 6, 4, 17, 4, 4, 4, 4, 6, 5, 4, 5, 5, 4, 6, 5, 6, 6, 9, 5, 5, 5, 9, 4, 3, 4, 7, 13, 7, 3, 4, 3, 4, 11, 4, 5, 4, 5, 4, 7, 3, 5, 4, 4, 5, 3, 5, 7, 5, 5, 3, 4, 4, 4, 4, 9, 6, 4, 4, 4, 6, 5, 6, 6, 4, 5, 7, 4, 5, 4, 5, 4, 5, 5, 4, 5, 4, 5, 5, 4, 5, 5, 5, 4, 8, 5, 6, 4, 4, 4, 5, 5, 4, 10, 4, 4, 4, 8, 5, 6, 8, 4, 11, 3, 6, 5, 4, 4, 4, 5, 4, 6, 4, 5, 4, 5, 4, 5, 4, 5, 5, 5, 7, 5, 8, 4, 6, 10, 4, 3, 6, 6, 5, 4, 4, 7, 8, 4, 5, 7, 4, 4, 7, 5, 4, 4, 5, 7, 6, 5, 8, 4, 4, 7, 5, 6, 5, 8, 4, 5, 6, 5, 6, 3, 4, 4, 4, 8, 3, 4, 9, 7, 4, 4, 4, 5, 6, 5, 4, 8, 5, 5, 5, 7, 4, 4, 5, 4, 4, 5, 5, 9, 4, 5, 4, 36, 6, 4, 5, 4, 5, 4, 4, 6, 5, 4, 4, 5, 5, 7, 4, 4, 6, 4, 5, 4, 4, 5, 4, 5, 4, 5, 4, 4, 5, 6, 5, 4, 5, 14, 6, 7, 5, 4, 6, 4, 11, 3, 4, 4, 4, 5, 8, 3, 5, 4, 5, 13, 8, 4, 5, 4, 5, 5, 3, 4, 4, 5, 4, 5, 4, 5, 4, 8, 6, 7, 4, 4, 10, 4, 4, 4, 4, 5, 5, 3, 4, 4, 3, 4, 5, 4, 4, 9, 5, 6, 3, 4, 6, 4, 4, 3, 4, 4, 5, 6, 4, 5, 5, 4, 4, 3, 5, 5, 5, 4, 6, 4, 5, 11, 7, 5, 3, 5, 5, 4, 6, 5, 5, 8, 4, 5, 5, 6, 6, 3, 5, 6, 4, 6, 4, 4, 4, 4, 5, 6, 4, 4, 4, 4, 4, 5, 6, 4, 4, 4, 6, 5, 4, 6, 4, 6, 3, 4, 4, 4, 5, 4, 4, 5, 4, 8, 4, 4, 12, 7, 4, 6, 5, 5, 4, 4, 4, 5, 5, 8, 4, 4, 4, 4, 4, 5, 4, 5, 4, 4, 6, 6, 4, 5, 4, 5, 4, 5, 4, 4, 4, 5, 4, 4, 6, 4, 6, 3, 5, 6, 4, 4, 4, 6, 5, 6, 4, 4, 5, 3, 4, 5, 5, 4, 4, 5, 5, 5, 4, 5, 4, 6, 7,

Total Time: -1474810645 milliseconds

**Server Router**

The Server Router for this project is the actual server. It’s the most complex part out of this project thus far. It contains Distributed Computing technology along with threading. Some key components are:

* ServerSocket: This is one of the biggest differences in the server router compared to the client/server technology. The ServerSocket is what makes this a server. It waits for requests to come in over the network. Then performs actions based on the requests sent to it.
* Client Socket: The server router will use a regular client socket just like the client/server. Once the server socket accepts a new connection we set our client socket equal to it. We then pass this socket information into the new communication thread that is created. It is then used for the forwarding of messages to the designated client/server.
* BufferedReader / PrintWriter: These two are used to read incoming messages from the socket and then write output messages through the socket.
* Communication Thread: This is the magic of the server router. The communication thread handles the actual forwarding of messages in its own independent thread.
* 2D array RouteTable: This array is used to store the routing information. It stores forwarding IP and Socket.

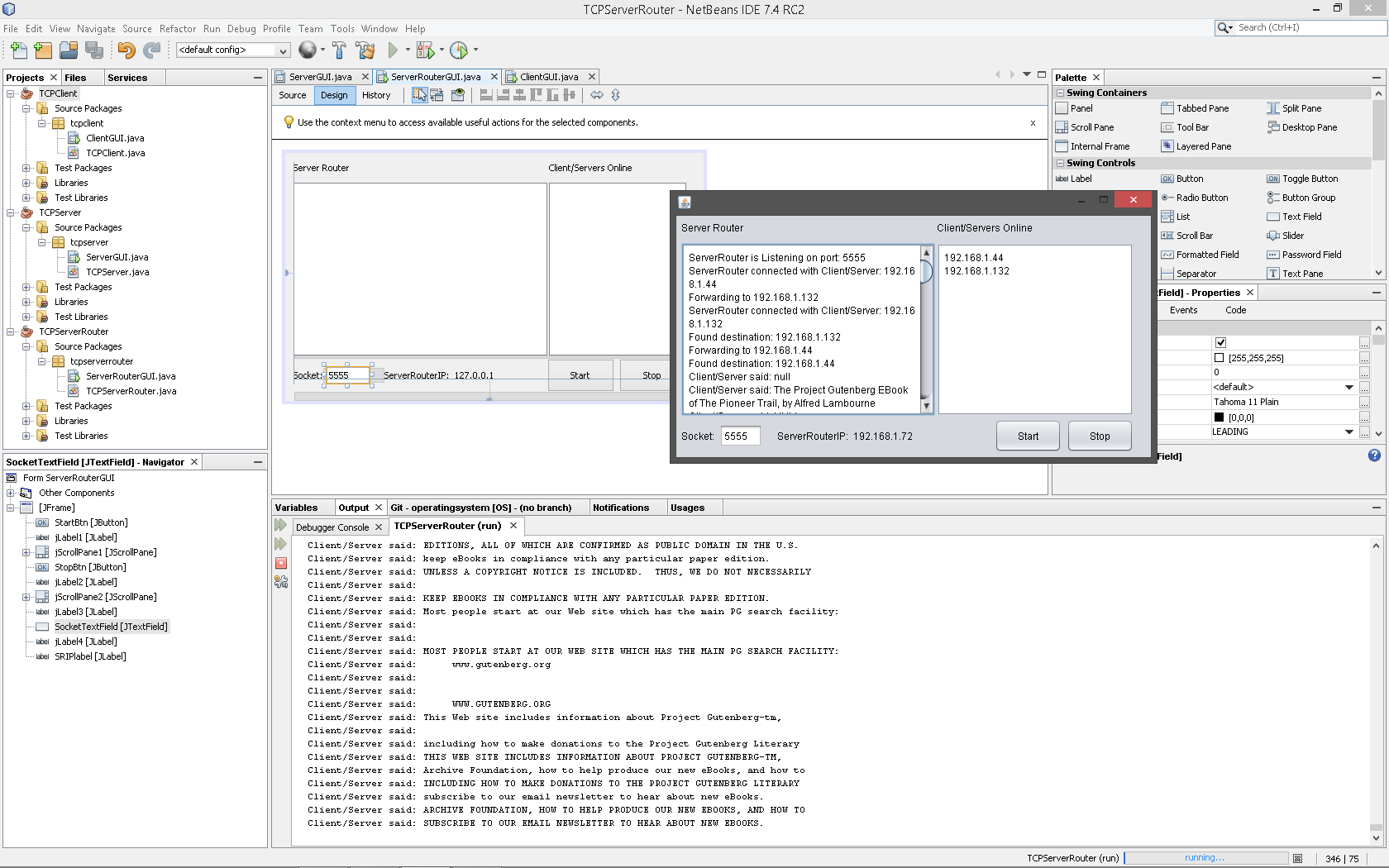
The server router started out as a simple then became a date time server, then turned into chat server, then evolved into what it is now. When it was a date time server user would send a message “DateTime”. The server would take the message, read it, look to see if it had a method to handle it, which it did, then send back the date and time to client. When it was a chat server it would just post the messages passed to from the client to a text box area.

For this phase of the project the server router needed to act as a middle man between the client/servers connecting to the server. We created a server socket that will listen on specified port. The user has the ability to set the port number before launching the server router. Also you can see the address of the server router at the bottom of the screen. This is helpful for when setting up the client/server. Once you click start the server router goes into action. We take in the socket being passed in from the GUI and instantiate the server socket with the socket number. Then placed inside a while loop we tell the server socket to listen for incoming connections. Once a connection is received we set out client socket equal to the socket information, and then create a new communication thread to handle the communication. We pass in the route table, client socket, and table index to the communication thread. After thread is created move real quickly to post a message saying who connected, then add them to the client/server list on the GUI. Then the program goes back to the top of the while loop, and if running is still true the server goes back into listening mode.

We’ve just created and started a new communication thread. This is a very important part of a multi-client to server distributed system. It allows the channel of communication between two clients to run independently and not be interrupted or blocked. The communication thread has its own private set of tools it uses. Similar to the client and server set up it has your main components:

* 2 PrintWriters, one for each client speaking with each other. One that send to destination specified and one that responds to the requestor.
* BufferedReader to read incoming messages.
* Another Socket that is used to send to the designated receiver (destination IP and Socket).
* 2D object Array that is passed in when creating the communication thread. This is what keeps track of senders and receivers and where messages are coming from and going too.

Everything else supports these components. In the constructor of this class we set one of the print writers equal to the pass in sockets output stream. Then set our buffered reader equal to the client sockets input stream. The after we set up our router table we enter in the client address and socket into the table. Now that we are set up, it’s time to Run. In the Run, the program first waits to receive the destination IP from the client/server for forwarding address. Then it sends back a response saying that the client/server has connected successfully. Then a delay was added to insure enough time was given for both client/server to connect and set up. If the receiver didn’t have enough time to get his information into the table the program would not be able to find the forwarding address. After the time is up, we iterate through the route table searching for the destination IP. Once found, we set our outgoing socket equal to the information from the table (IP and Socket). Then we set the outgoing print writer equal to the outgoing socket output stream. Now that the link is created and we have out communication portal the server router can pass messages back and forth. This all happens inside the communication loop. While data is coming into the server router from the client/server it will forward it to the destination that is specified.



ServerRouter:

ServerRouter is Listening on port: 5555

ServerRouter connected with Client/Server: 192.168.1.44

Forwarding to 192.168.1.132

ServerRouter connected with Client/Server: 192.168.1.132

Found destination: 192.168.1.132

Forwarding to 192.168.1.44

Found destination: 192.168.1.44

Client/Server said: null

Client/Server said: The Project Gutenberg EBook of The Pioneer Trail, by Alfred Lambourne

Client/Server said: NULL

Client/Server said:

Client/Server said: THE PROJECT GUTENBERG EBOOK OF THE PIONEER TRAIL, BY ALFRED LAMBOURNE

Client/Server said: This eBook is for the use of anyone anywhere at no cost and with

Client/Server said:

Client/Server said: almost no restrictions whatsoever. You may copy it, give it away or

Client/Server said: THIS EBOOK IS FOR THE USE OF ANYONE ANYWHERE AT NO COST AND WITH

**Server**

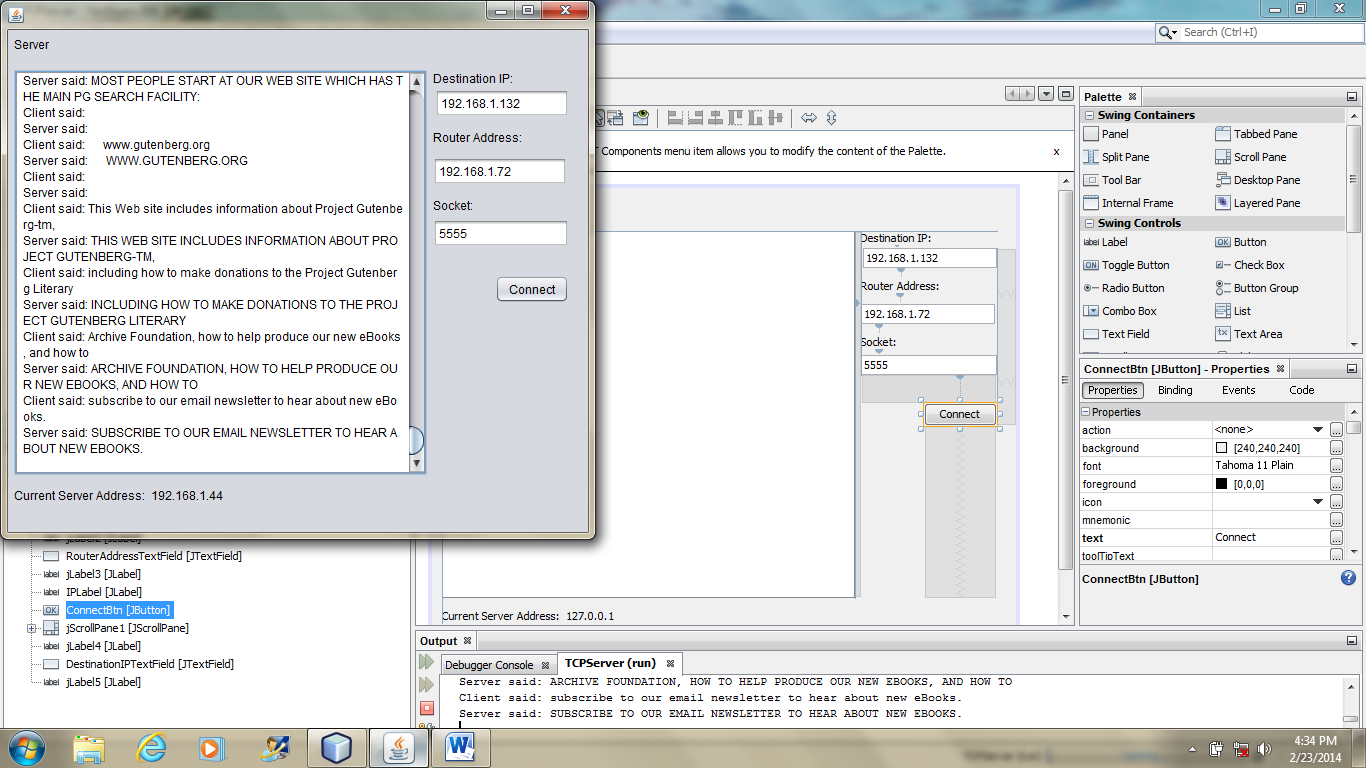
The server side of this project as of now is very similar to the client. The main difference is it takes in the text being sent from the client, capitalizes it and then sends it back. The components are:

* Stream Socket: this is the client socket that will be used to send and receive the data to the server router. It points towards the IP address and socket of the server router
* PrintWriter for writing into the stream
* BufferedReader for reading from the stream
* Method to capitalize the text

This was the easiest piece of the project to implement. The server would simply read in from the input stream, take the text passed to it via the server router from the client, capitalize it, and then send it back through the output stream of the socket.

In the design side of this started with a basic client set up. Created a stream socket with default properties and set the IP and Socket to use. Tested it first with simple text messages and made sure it could send and receive. Next we added in the capitalizing feature of the server which took the text and capitalized it then the program would send it back across the output stream. I then wrapped it in a GUI like the rest of the program which made it easier for a user to set up before launching. The server runs the same as the client; it starts by taking values given to it from the GUI and sets up its connection to the server router. Once user clicks connect the program sets the values for the socket (routerIP and Port Number). Then it sets the printwriter to the output stream for writing and the bufferedReader to the inputstream to read incoming messages. The program tells the user what’s going on in the message area of the GUI. Then the server sends out its initial send to the server router letting it now the destination IP it wants to send too. Now the program waits for the response before moving on. Once the initial receive comes from the router verifying it has a connection it waits for the client to send it some text to capitalize. As long as the client is sending text the server will continue to loop in a while loop executing the capitalizing method and outputting it back into the stream to the router then to the client. The text returning to the client will be in all caps.

Server run:



Trying to connect. Router: 192.168.1.72 Socket: 5555

Connection completed.

ServerRouter: Connected to the router.

Client said: null

Server said: NULL

Client said: The Project Gutenberg EBook of The Pioneer Trail, by Alfred Lambourne

Server said: THE PROJECT GUTENBERG EBOOK OF THE PIONEER TRAIL, BY ALFRED LAMBOURNE

Below is a link to a live demo using a 138kb plain text file:

<http://www.youtube.com/watch?v=yhHVWnOX0r0&feature=youtu.be>

In conclusion this project helped us gain an understanding of a basic distributed computing application. We now know how to set up sockets, different types of sockets, and how to use these sockets. We learned how to implement thread control that can help handle multiple connections and processes. There is much more to learn and there is more we wanted to implement just didn’t get the time too. It would be nice to be able to launch the client and have the server automatically detect it and add it into the client list. Also wanted to add in a client list into the server and client so then the user could select by clicking who they wanted to forward their message too. We are happy with the outcome of phase I and are excited to see how phase II will challenge us.