Josh Bucklin
CS300
TauNet – Project Evaluation Report

Overview & Approach:
Let me start off by saying I think this was the most enjoyable project so far in the CS curriculum. I feel that both creating and documenting the project allowed me to gain experience that is applicable to the real world. Creating the project also provided me with challenges that I had not yet encountered such as dealing with sockets or following a protocol. My initial confusion about these topics led me down some exploratory avenues which helped me to achieve my final result.

There are actually two versions of TauNet that I created. The first one (which is near the beginning of my repo and doesn't work that well) combined both the server and the client into one program. I think my natural inclination was to create it this way because that's what I had seen other people do (AIM, Yahoo messenger, etc...) However, for this project I don't think that is the best approach. The first implementation was clunky and forced the user to constantly check an inbox system to see if they had any new messages. I knew immediately that I wanted to simplify the design so that I wasn't saving messages and the process of reading messages was more streamlined.

Then I thought of an approach to simplify the program by splitting the server and client into two applications. I think that given the nature of the project and the need to use Linux terminals it works very well. You can simply open two terminals and load each with a client and server and you have a working chat system.

Once the server was its own program it allowed messages to be received and displayed instantly. I still added a menu functionality to the program because I think it's nice to have things like help menus if needed. So I was still faced with the task of managing output from multiple threads. Initially I was intimidated by it but after Bart's lecture on locks I realized how simple (is a lock ever simple?) it could be. Another reason the menu is valuable is that it allows the user to close the server with a command instead of just closing the terminal and forcing the program shut.

The client portion of my code retained much of the same flavor as the first iteration that I created. I had considered just making a terminal like program with a prompt instead where the user can just type a username and message into but I thought that would be a little vague. I'm really a fan of menus in programs because it guides the user directly to what they should be doing at each point.

Another functionality that I took the liberty of adding was the ability to change the encryption key during a session of TauNet. I'm not a big fan of having the key stored locally in a text file because that could be easily compromised. I instead focused on having the user decide what the key should be each time the program is started and gave them the ability to change it without restarting the program. It worked out pretty well and provided a good way to test my locks in the server.

One drawback to this approach is that if you are attempting to use this with ssh you either have to open two ssh connections to see both windows or you constantly have to switch between them. However, I don't really think it's that heavy of a drawback to justify changing it.

Testing:
As I was creating the test plan I actually made notes on all the tests that I finished and how they went for me. As a result most of my information about testing can be found in the STP document. My main focus during testing was to make sure that all the SRS requirements were met. It actually ended up working out pretty well this way because I kept my initial requirements to a minimal set and I only ended up needing a small amount of tests.

I also made a small testing program for cs2 that just runs a bunch of input through encrypt and decrypt and makes sure the results are what is expected. Again, this is talked about in the STP document.

Probably the most helpful testing that I did was making sure my TauNet worked with other nodes. I was able to successfully test with numerous users in the class as well as Bart's echo server. The bulk of my testing with other users was with Eric Tsai. We messaged each other on a few different occasions and it helped to spot minor errors that we both had made. After the echo server was live I focused most of my testing on that but by that point my implementation was mostly working except for a few header errors. Screenshots of these sessions are provided.

In addition to simply sending messages back and forth Eric and I both reviewed each other's code. This was a helpful process to see how someone else went about implementing a project like this. I think it would actually benefit everybody in the class to have the opportunity to look at other people's code. It often feels like we are coding solo in a class like this but in the real world we will be subjected to other people's opinion's constantly so it would be nice to have a head start.

In the end all of the tests that I wrote I was able to pass so I was satisfied with my implementation.

Errors, Changes & Speed:
There are a few improvements that I would like to see made in my project.
The first improvement that I would like to see made on this project is the ability to see messages that I have sent to other users. While I was testing I constantly forgot what I wrote to who and it made it complicated to have multiple conversations happen simultaneously. An idea I have about how to improve this is to save the sent messages locally and then when a user responds, both the sent message and the received message are displayed. I would then delete the saved message to avoid overly large amounts of data being stored.

Another area that I would like to see changed is the how the incoming messages are checked to see if they are from a known user. The data structure that I used to implement this feature is a list of IP's that is checked each time a message is received. The creation of this list is the slowest part of my program and causes the program to hang for a few seconds when the program is started. This happens from having to use the socket.gethostbyname() function. Around 20 user-names must be resolved and the program pauses while all the IP's are located. It's ultimately not very noticeable but if you were to type a command right when you opened the program it takes roughly 4-5 seconds to get the first response. I really didn't see any way around this because I wanted the list of IP's loaded right away so if they failed to load the program would crash right away.

I don't like how my user name is also included in the user table but to remove it just seemed pointless and a waste of resources. It always fails if you try and send a message to yourself so I figured if the user wanted to try and do it they should be allowed but shouldn't expect too much.

One last area that I struggled with for a while and eventually just gave up on is to make sure the server

starts before the menu loop is started. If two versions of TauNet are started on the same machine the binding fails when the server is created because the port is already being used. And because the main loop is in another thread it causes the user to have to close the failed program themselves. I'm really not a big fan of this. I attempted to use thread locks in all sorts of creative ways but it just got insanely complicated. I found the code was more readable without worrying about it and instead just used an assert statement that crashes the program if the server doesn't start. However, even if the assert was disabled the worst that would happen is the user has to close the program themselves and restart. No big deal.

Closing Thoughts:
I learned a very valuable lesson about testing from this project. I think you should always be developing tests for your code as you are developing your code. I was panicked towards the beginning of this project because I didn't know much about the subject matter (TCP, Protocol, etc.) so I just dove straight into coding to get something rolling. After I did this though I found it was more difficult to go back and write tests for my code because I wasn't in the same frame of mind as when I wrote it. Lesson learned in a big way.

If I had more time with this project I might try and implement something with the python module curses to see if I could make a really cool interface. Overall I am satisfied with how my implementation turned out. I'm also glad I learned to use git. Thank you!