Team Assignment 4
CDF Analysis

Denver Hoggatt
Spencer Clegg
Colton Ottley
Rohith Prasad

Assignment 4 asked us to create a BeagleBone Black server-client model, where the client would send a string to the server, and the server would decrypt the string using ROT13, and send it back to the client. All of this was timed, and run multiple times. After many trials, the results were collected, compiled together, and then used to generate a simple Cumulative Distribution Plot.

First, we implemented the server-client on two BeagleBone Blacks, which would communicate over a serial line. Our client simply stored a string, which in our case we used "Hello World", encrypted it using a ROT13 algorithm, and then started a timer. The client would send the message, read whatever value was returned, stop the timer, compute the difference between the two times, sleep for 100 milliseconds, and then repeat. We would do this for a full 2-minutes, pipe all output into a text file, and then use this to compute our CDFs, which was done using a simple MATLAB script.

The output of our function is shown in figure 1:

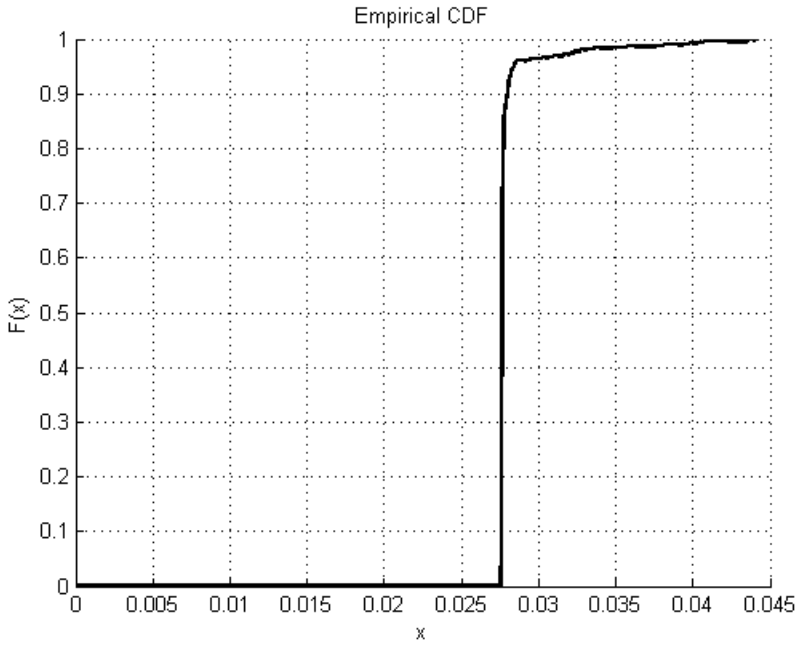

Figure 1: Cumulative Distribution Function of the Measured Execution Time

Next, in order to illustrate the effect of timing and CPU constraints on a program, we ran our program in parallel with an Iperf program reading network calls over the BeagleBone Black's standard USB input. The output of this run can be seen in figure 2:

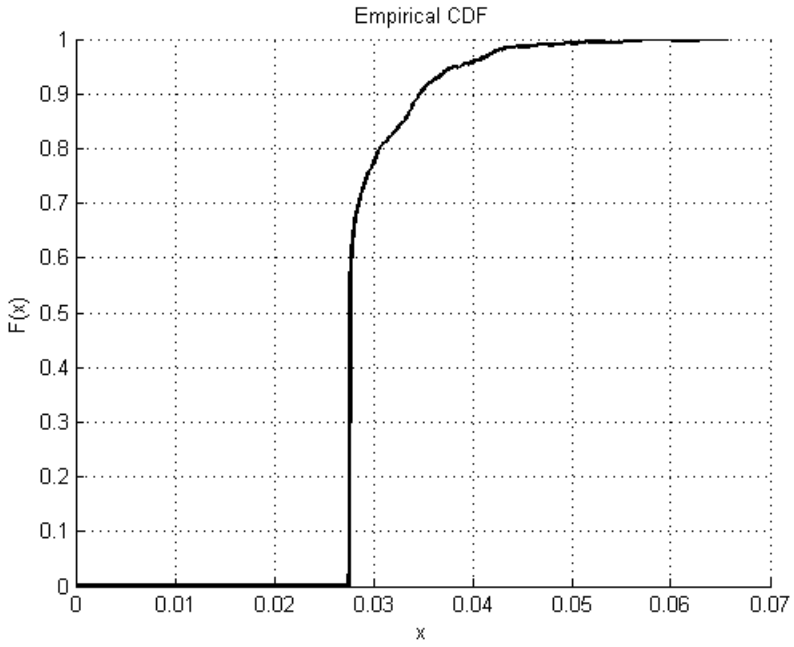

Figure 2: Cumulative Distribution Function of the Measured Execution Time with Iperf Running

As can be seen, the Iperf process adds a significant amount of slowdown and stochasticity in our progam. Thus, we were given the task of speeding up our program as much as possible. One simple way to accomplish this is to run the program in a higher priority than the Iperf function. However, to illustrate the effect of priority, we ran our program with a lower priority. The results can be seen in figure 3:

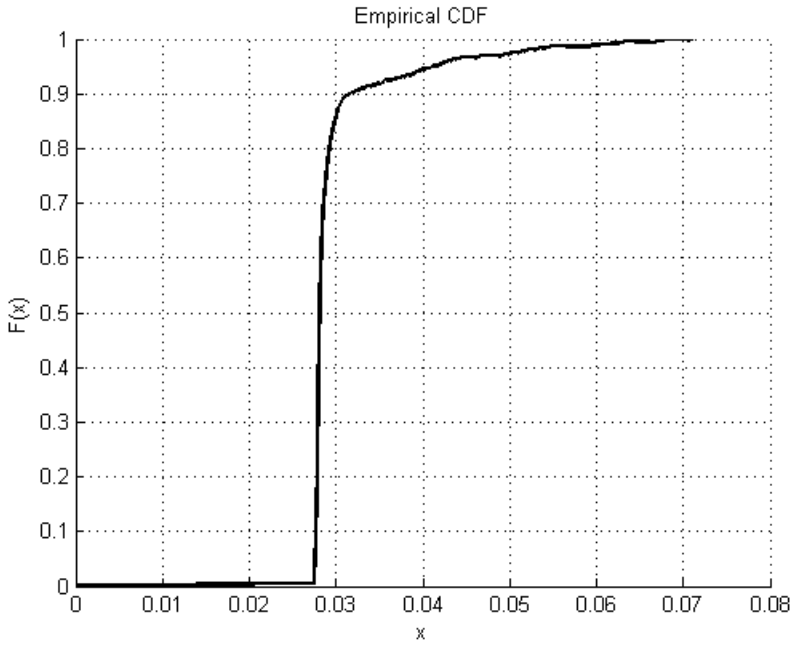

Figure 3: Cumulative Distribution Function of the Measured Execution Time with Low Priority

Running the program in a higher priority gives us the following result:

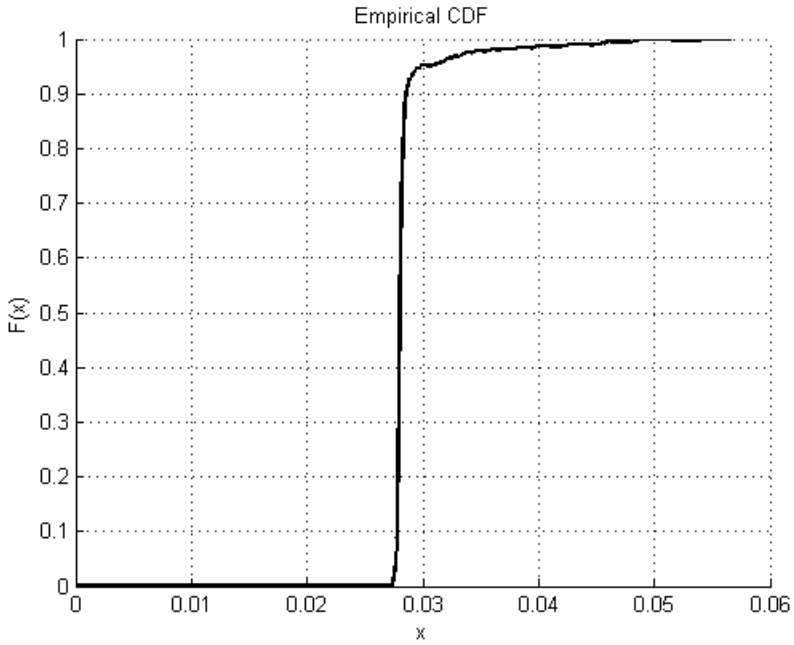

Figure 4: Cumulative Distribution Function of the Measured Execution Time with High Priority

As can be seen, a higher priority gives a significant speedup. To even further increase this, we tried running our program one of the standard PRUs that accompanied the Beagle-Bone Black. However, getting this PRU to correctly run the code proved challenging, and significant changes to the original code were made. As such, the results, as seen in figure 5, are to an extent unreliable, since the exact same code was not used.

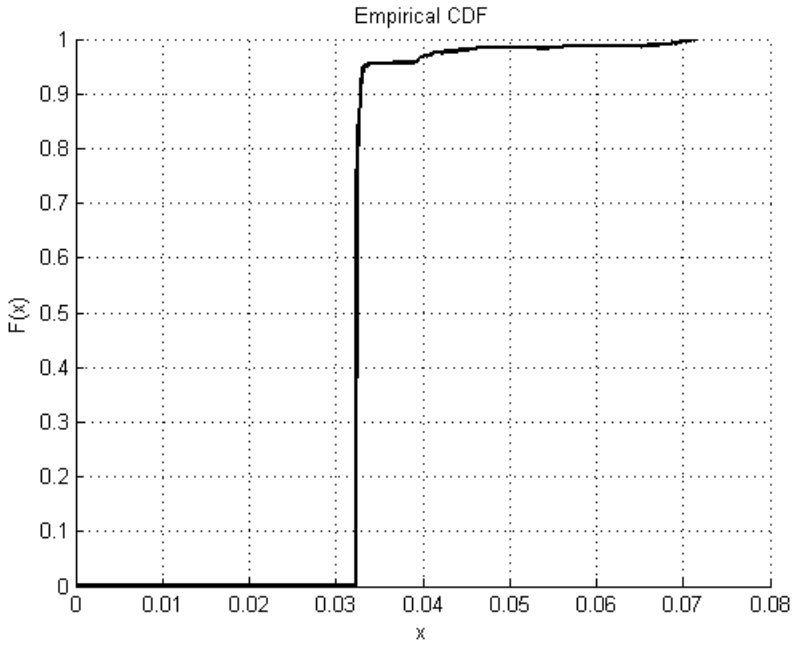

Figure 5: Cumulative Distribution Function of the Measured Execution Time with Iperf in Low Priority and Program in High Priority

Finally, we compiled all of the CDFs into one single plot to examine the differences between each of the functions.

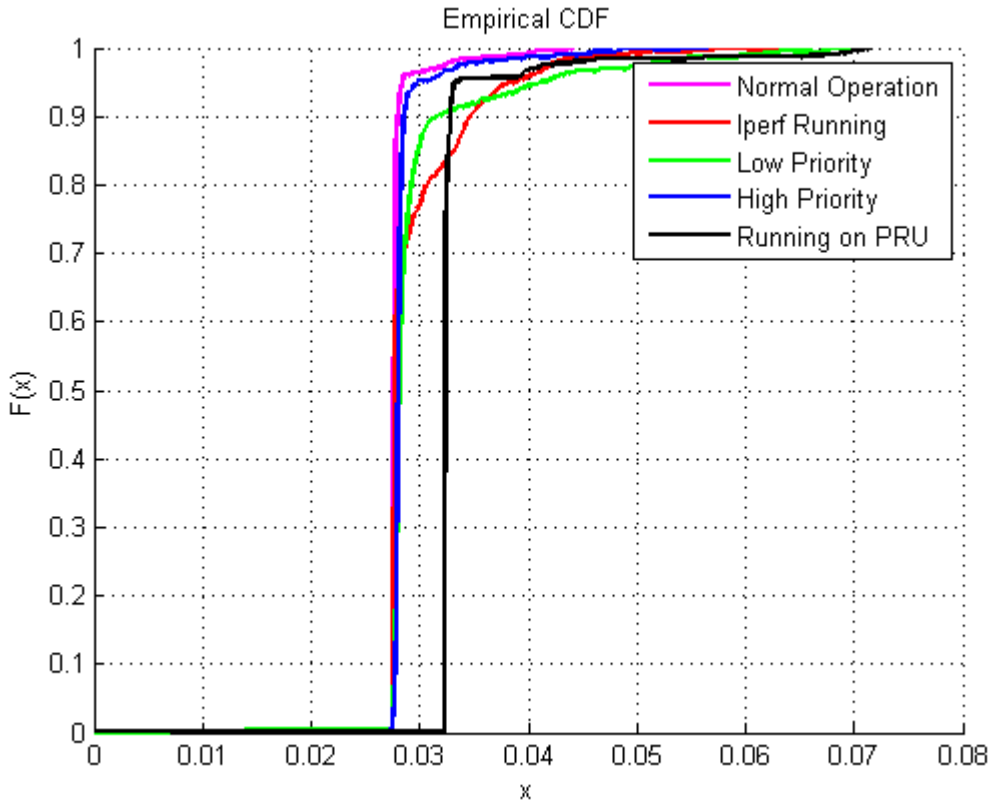

Figure 6: All Cumulative Distribution Functions Together

The first thing to note is that the normal operation operates both in the lowest amount of time and with the most reliability. However, running our function by simply increasing the priority such that it is higher than all other interfering function gives us a result almost as fast and almost as reliable, especially when compared to normal Iperf operation, or running the program with a low priority. Lastly, attempting to run the code on a separate PRU showed better reliability than that of a high priority operation, almost identical to that of normal operation, but it consistently ran at a slower pace.

There are several reasons why this might be the case. First, in order to compile the code, a whole different tool-chain and compiler were used to generate the client code. While getting this code working, major modifications were made, and it is not certain that the code was exactly the same. Beyond this, the compiler could have compiled the code in a less efficient manner than the original code. Lastly, the subsystem used for the PRUs are likely significantly slower in actual code execution, since they are separate from the 1 GHz ARM core used on the BeagleBone, and likely have lower execution speeds. Nonetheless, simply increasing the priority of the function seemed to produce better results.

If given more time, we would have tried to clean and optimize the code used for the PRU execution so that we were certain the code was as efficient as possible. If this sill was not acceptable, we would have tried to code and load a kernel module that would execute our program, as this would even further decrease the chance that our program was being interrupted by Iperf and other programs.