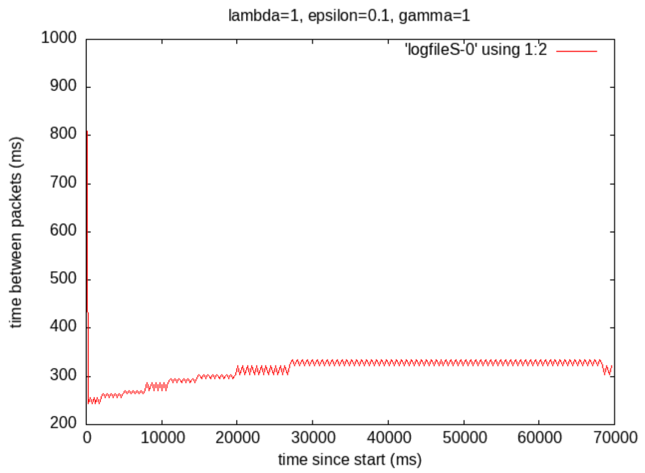
Jimson Huang and Stephan Zapodeanu
We will use 1 late day for this assignment

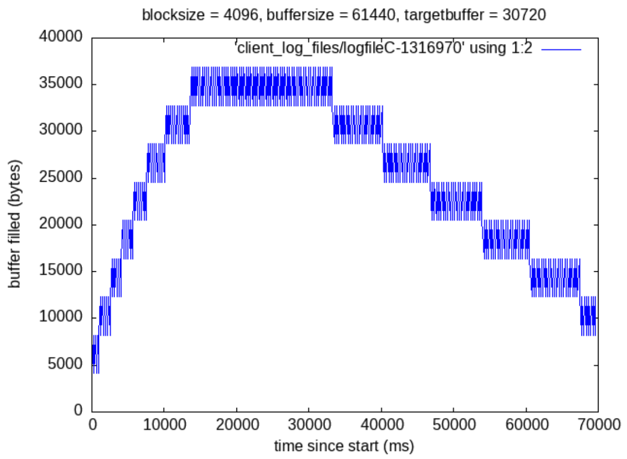
Work Distribution: For this lab, we split up the workload mainly on client and server. Jimson mainly worked on the client while Stephan did most of the server. We worked on this lab at the same time in HAAS g50 so we could each discuss the best way to set up the server and client relationship. Jimson also wrote the gnuplot code to generate the plots and Stephan wrote the README as well the comments on the report.

All plots are on pp.au. The plots in red are for the server, and blue are for the client. We deleted all plots and log files from the turn-in directory prior to turning in this assignment

Method D: For the plots for method D, we spent time trying to find a combination that would converge for pp.au. Then, we changed each variable to see how altering that variable had an impact on the generated plots:

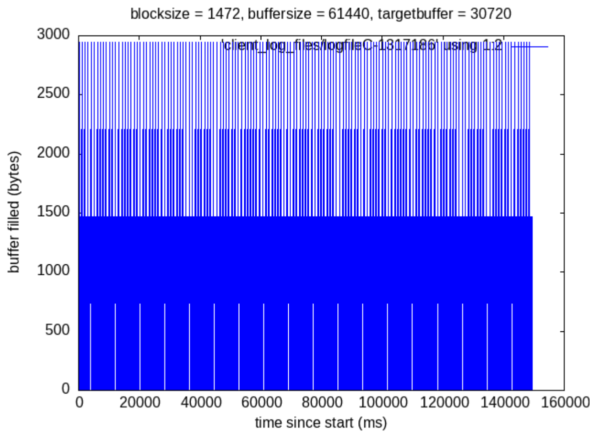
Plots 1 (control):
Command Line Arguments:
Lambda: 1, Epsilon: .1, Gamma: 1, blocksize = 4096.
We found that if epsilon was too large, then the bufferstate received from the client would be too volatile. Thus, we shortened it so that lambda would change at a more gradual rate. We found that if this value was too high lambda would become negative (or .6 as we set it too to ensure that one packet would get sent at least every 2 seconds).

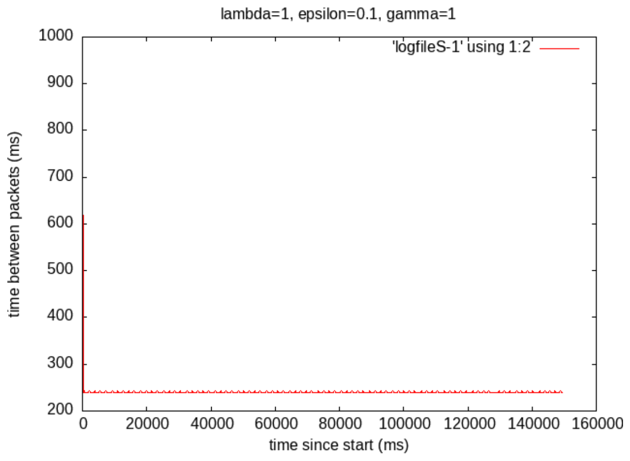




Plots 2:
This was for when we reduced the block size of one ethernet packet, as suggested in the handout. As can be seen, no semblance of convergence occurs for the client and the server essentially sends data as fast as possible. This is because the packet size is much smaller, so

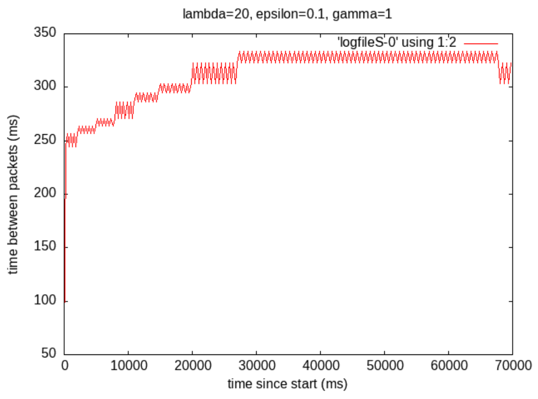
the more need to be sent to fill the buffer. We will include a different plot with the same block size later on.

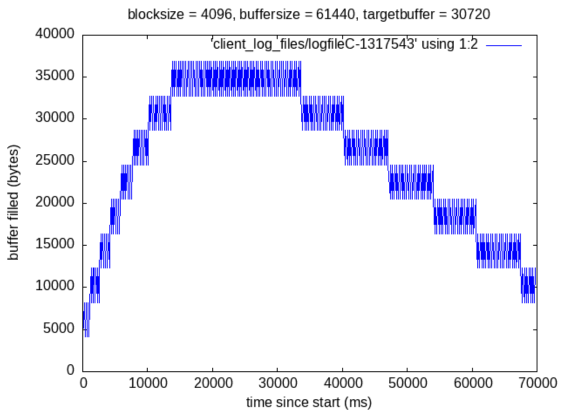




Plots 3:

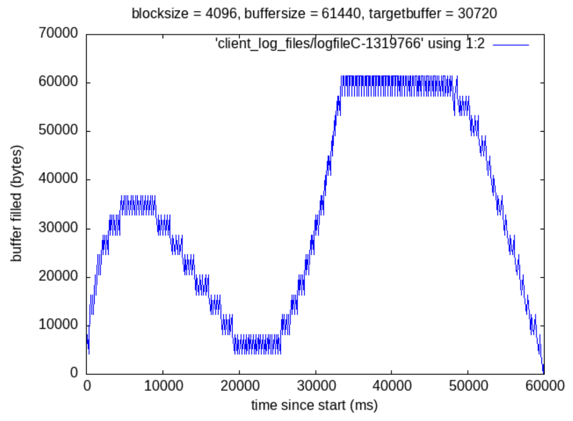
This plot is the same as the first one, but we wanted to see how the original plot size would change how the server plot would look. As can be seen from the y-axis, the time between packets converged to around the same value but came from opposite directions. Because the lambda here started much higher the rate of the first couple of packets were much much faster than those of plot 1. This, would be a better solution for the server as it would not need to ‘speed up to keep up with demand like that of plot 1.

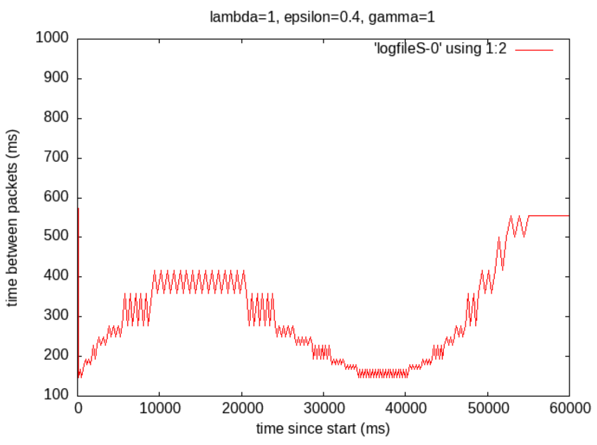




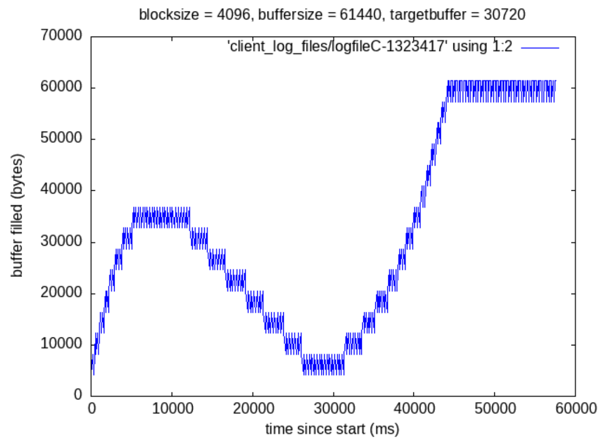
Plots 4:

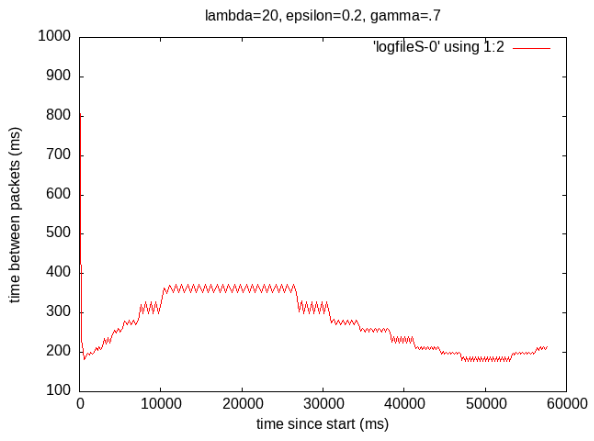
In the following plot, we increased the size of epsilon from .1 to .4. As can be seen, this caused each time between packet to be higher. We believe this is because increasing the epsilon value would cause the lambda value to become much more volatile which would cause it to sleep much longer than what the client would need. The client buffer was empty for about half the file transfer, but then filled up.



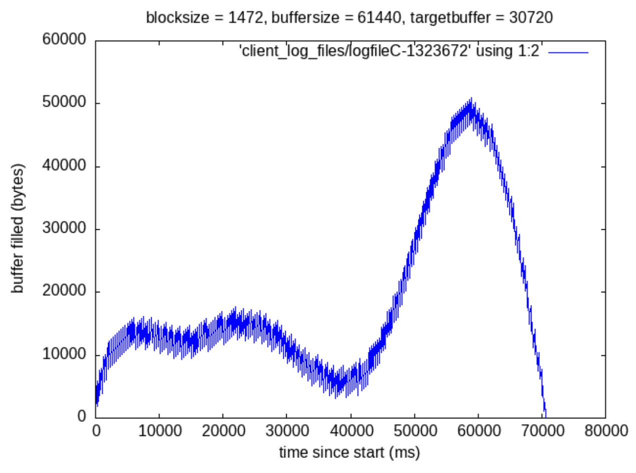


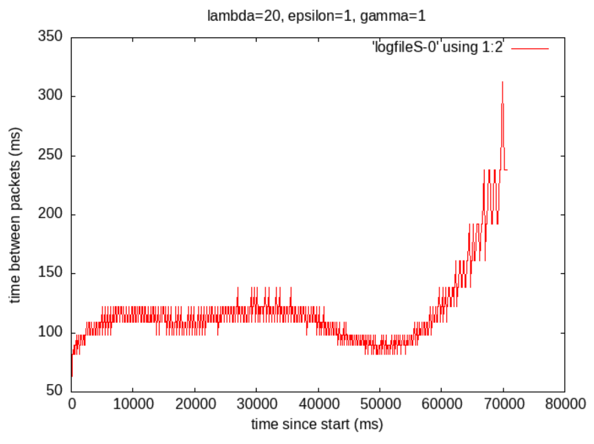
Plots 5:
For this one, we changed the gamma value (as well as slightly increase the epsilon to make up for it). There was some fluctuation, but overall the client buffer didn’t reach full capacity until the end.





Plots 6:
The following plot, we wanted to get the block size to be an ethernet packet, but fill the buffer up (in a gradual way). As can be seen in the graph, for most of the data transfer, the buffer was not full. Also, it appeared that the server was sending at a constant rate to the client until the very end (when the client’s buffer filled up)

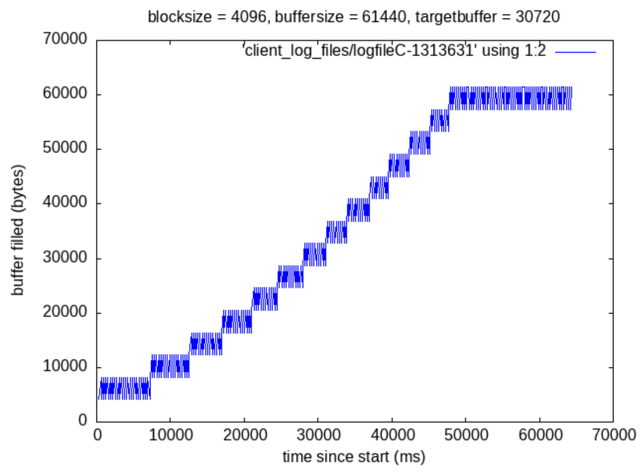


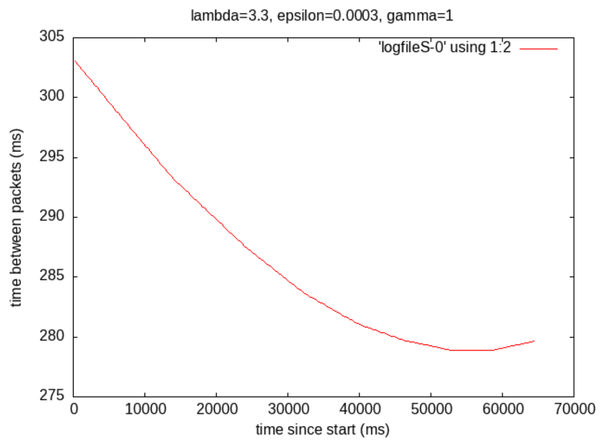


Method C:

For this method, we used pp.au as well and followed a similar approach as in method d to generate plots (although there are fewer variables so we only made 3 plots)

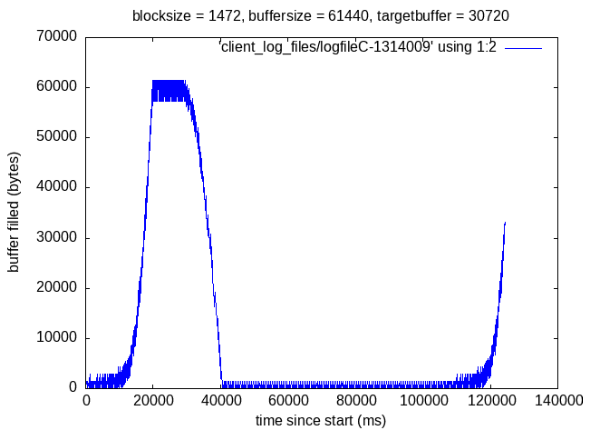
Plots 1:
After testing, this was the best combination of Command Line arguments that made the server plot ‘converge’. We had to set lambda to very close the ideal sending rate and epsilon very small. This was because in method c that the client buffer would constantly fill up and the server would not be able to slow down enough such that the client can eat up some of the buffer. This was rather surprising considering our 0.6 minimum lambda value. Overall, it can be seen that the lambda value does constantly decrease (from the server plot), until it finally gets the feedback from the server right at the end and then begins to slow down.

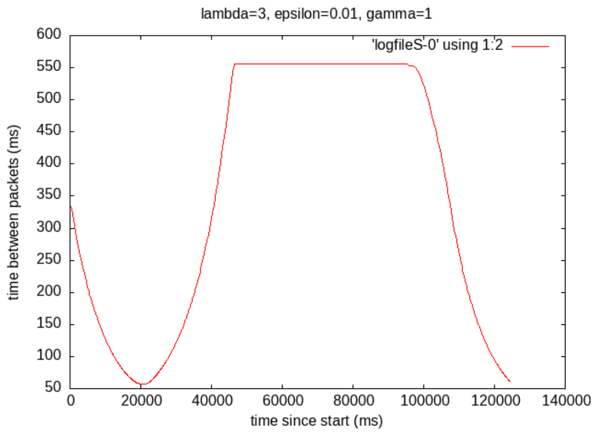




Plots 2:

For these plots, we decided to increase the epsilon from lambda as well as decrease the block size to see if the buffer could be filled. As can be seen from the graphs, the buffer ballooned up to the maximum capacity until the sending rate decreased and was never able to recover. Perhaps due to the epsilon values still being very small and not being able to increase in time.


blocksize = 1472, buffersize = 61440, targetbuffer = 30720
'client_log_files/logfileC-1314009' using 1:2
buffer filled (bytes)
time since start (ms)


lambda=3, epsilon=0.01, gamma=1
'logfileS-0' using 1:2
time between packets (ms)
time since start (ms)