Latency Mitigation

Based on the previous graphs when we first implemented latency, we can conclude that excluding natural latency across servers, we have a relatively stable graph with execution time on the x-axis, and estimated latency(in milliseconds) on the y-axis that follows our expectations of how the latency should be reflected on a timed interval.

Due to how we structured our latency buffer, we decided that it would be more fitting that we use Time Delay Model instead of Time Wrap model. How we created latency was that we created a queue on the server side that only updates game state at every designated server ticks through periodic handler. [add one more or two sentences]. After we implemented the mitigation mechanism, the periodic handler will now take a boolean parameter that determines whether the server updates game state, and how long should the server delays itself to update the game state. After we’ve implemented the latency mitigation mechanism, we started the testing process by connecting to the server from two more different machines and stream on one of the machines to create the heaviest workload on one machine. We then record the latency value on the client and compare the resulting graph to our initial latency graphs from Milestone 5. The results are shown below:

[insert graph]

Result shows that the ping values per tick was lowered on an average of [number] ms. We had successfully mitigated part of the latency we’ve created in the first place.