Discrete Event Simulation: Call Center

Felice Chan and Jui Chang Lu
(fchan9 | jlu345)

October 13, 2017

1. Introduction

Discrete event simulation is commonly used in science and engineering applications to analyze a system, which can be anything from a gas station (as it was in the sample simulation) to an airport. In this case, discrete event simulation is being applied to a call center that places both receives incoming calls and places outgoing calls, just as a realistic call center would.

For the purposes of this assignment, the call center had a total of 1,000 employees that handle phone calls: 700 service agents that could only handle incoming calls, and 300 sales agents that were able to handle both incoming and outgoing calls. The call center would first route the incoming calls to service agents, and if none are available, then it would route the calls to sales agents, and if no sales agents were available, 25% of the callers would hang up, and 75% would wait in a queue. In this assignment, we assumed that once a caller entered the queue, he/she would not leave. The service and sales agents would then answer the calls from people in the queue in first come first serve order.

To conduct business and sell more product however, the call center would automatically place a number of outgoing calls every 60 seconds, and after a 10 second lag, the call would be routed to a sales agent who would hopefully, answer the phone. If no service agents are available, the call would be dropped. To complicate things further, the system does not know how many outgoing calls to attempt, and therefore estimates the number of calls to make based on the probability that the calls were answered in previous outgoing call scenarios. The simulation is modeled for 8 hours, simulating the length of an average workday.

2. Description of Application Program Interface

On top of the application program interface (“sim.h”). There are two additional function: random_number generator, as it was requested on the assignment. The other one is init_queue. Since simulation application is the script that contains the main function. It is necessary for the simulation application to call the function to initialize the main array and the bucket list subarray.

3. Implementation of Simulation Application

To make the process easier to understand, the pseudocode is split into two sections: Incoming calls and outgoing calls.

3.1 Pseudo code for Simulation Application – Incoming Calls

For the incoming call simulation, there are several things that we need to keep track of: The total number of service agents on the phone, the total number of sales agents on the phone, the number of people currently in the queue and the time they entered and exited the queue, the total number of people that were ever in the queue, and the number of people that did not have to wait in the queue. The function also calculates the total number of incoming calls so that percentages can be determined later.

For the implementation of the Incoming calls, two event types are defined: Incoming, which is the incoming call event, where the agent picks up the phone, and Finishing_Inc, where the agent hangs up the phone.

First, a structure for event data is created which monitors the event type, and whether the type of agent that picks up is a service agent or a sales agent.

Incoming Call Function:

- First this schedules the next incoming call event, modeled on a Poisson Distribution. Increment total calls.
- Then the function checks if there is anyone in the queue. If there is no one in the queue:
    - Check if there are available service agents. If yes, then route the call to the service agent (increment ServiceOnPhone), and schedule the end of the call (Finishing inc). The end of the call is scheduled as a uniformly distributed random number from 300 to 700s.
        - Increment NoWaiting
    - Check if there are available sales agents. If yes, then route the call to the sales agent (increment SalesOnPhone), and schedule the end of the call.
    - If all sales and all service agents are busy, then the person has a 25% chance of hanging up
        - Generate a random number from 0 to 100, if the number is less than 25, then increment HangUp. If not, then they join the queue.
- Then the function checks if there are already people in the queue (InQueue > 0)
    - If all agents are busy

        - Generate a random number from 0 to 100. If it is less than 25, then increment HangUp. If not, then they join the queue.
    - If service agents are free, but sales agents are busy
        - The incoming caller joins the queue (increment totInQueue) and the service agent picks up the phone from the first person in the queue, increment ServiceOnPhone, and schedule a hang up event.
    - If service agents are all busy, but sales agents are free
        - The incoming caller joins the queue (increment totInQueue), and the sales agent picks up the phone from someone in the queue, increment SalesOnPhone, and schedule a hang up event.
- End Incoming Call:
    - If the type of agent hanging up is a service agent, decrement service agent.
    - If the type of agent hanging up is a sales agent, decrement sales agent.
    - If there are people in the queue:
        - If the type of agent hanging up is a service agent, they pick up the next phone call from the person in the queue, so increment ServiceOnPhone. Since the agent answered the phone from the queue, decrement the queue.
        - If the type of agent hanging up is a sales agent, they pick up the next phone call from the person in the queue, so increment SalesOnPhone. Since the agent answered the phone from the queue, decrement the queue.

3.2 Pseudo code for Simulation Application – Outgoing Calls

Three event types are created for outgoing calls: outgoing_eval, which evaluates the number of sales agents available and determines how many calls it should attempt, outgoing, which is the actual event where the sales agent picks up the phone, 10 seconds after outgoing_eval, and the finishing_out, which is the ending call event for outgoing calls. As was described in the problem statement, every 60 seconds the system tries to place an outgoing call based on the number of currently available sales agents.

Evaluate Outgoing Calls to Attempt:

- First this calculates the number of idle sales agents (total number of sales agents – SalesOnPhone)
- If the number of outgoing calls is greater than 0, calculate s_est = Successful outgoing calls / total outgoing
- Then this calculates number of calls to attempt, NsalesIdle / s_est

- Schedule the agents to pick up the phone in 10 seconds (Event type: Outgoing)

Outgoing Calls being Answered:

- Loop over the outgoing calls being placed by the number of outgoing calls determined in the previous function.
- If time is less than one hour or greater than 5 hours, S_true is 0.4. If time is between 1 hour and 5 hours, then S_true is 0.6.
- Generate a random number from 0 to 1, if the number is less than S_true, then increment unsuccessful.
- Else, the customer picks up, so the call is considered successful. Increment SuccessfulOutgoing. If there are no sales agents available (NsalesIdle = 0), then increment abandoned. If there are idle sales agents, then they answer the phone (increment SalesOnPhone) and schedule a hangup event, (sometime between 200 and 400 seconds later).
- Schedule the next outgoing_eval

Finishing Outgoing Calls

- Decrement SalesOnPhone

4. Testing Procedure of Simulation Application

To test the simulation application, the program was run with both the sample engine (sample code provided by Dr. Fujimoto), and my partner's engine. For both, the simulations were run for 8 hours (28,800 seconds), to simulate an average workday and the same rate parameter. The program should yield similar results from both engines.

The most basic check for correctness is to print the number of sales agents on the phone and the number of service agents on the phone for each time, and make sure that they never fall below 0, or reach higher than the total number of each type of agent.

An additional method to ensure that the code was producing the correct metrics of incoming callers, the total number of callers was incremented, and compared to the sum of number of people that did not wait in the queue, the number of people that hung up, and the number of people that waited in the queue, which should be the same number. In addition, the percentages of no waiting, hung up, and waited in queue, were computed and summed to equal 100%. Sample results of light, medium and heavy traffic for incoming calls are summarized in the following table. The rate parameter used in the poisson distribution is given in parenthesis.

Table 1: Incoming Callers Metrics with Sample Engine

	No Wait	**Hung Up**	**Entered Queue**	**Total**

Light (1.5)	99.391571%	0.147036%	0.461390%	100.0%
Medium (2)	82.826729%	4.524443%	12.64883%	100.0%
Heavy (5)	6.213882%	24.159753%	69.626549%	100.0%

As expected, the sum of all of these quantities equals 100%, meaning no incoming caller went unaccounted for. In addition, for each incoming call, a print statement is output, with the sum of these three quantities and along with a counter for the incoming callers, which outputs the same value in each step.

Outgoing Calls

To verify that the outgoing calls are being placed correctly, the number of total, successful, abandoned, and unsuccessful calls were tallied. The application was tested with the sample code to see if the number of successful and number of unsuccessful calls summed to equal the total calls. The table below shows the results of light, medium and heavy traffic for total outgoing calls.

Table 2: Outgoing Call Metrics with Sample Engine

	Total Successful	Total Unsuccessful	Total Outgoing Calls
Light (1.5)	23,987	24,778	48,765
Medium (2)	14,745	14,572	29,317
Heavy (5)	430	316	746

The number of outgoing calls is dependent upon the number of sales agents available and at a time 10 seconds prior to when the call will actually be routed to the agent, and the value of S estimate. The number of outgoing calls to attempt is equal to the number of idle sales agents, divided by the value of S estimate.

S estimate was computed as the number of successful outgoing calls divided by the total number of outgoing calls, as determined in the previous outgoing call event. To improve the system, and make it more robust, it would be better to take an average of the value of S estimate for a number of previous outgoing call events.

The figure below shows a graph of S true (0.4 for one hour, 0.6 after that and then 0.4 again after 5 hours), and the values of S estimate run for the sample engine. The value of S estimate was initialized to 0.6, and the graph decreases initially, as the value of S true was lower than that of the initial S estimate, but then increases eventually to follow the increased value of S true. It does not have enough time to decrease again to a closer value to S true.

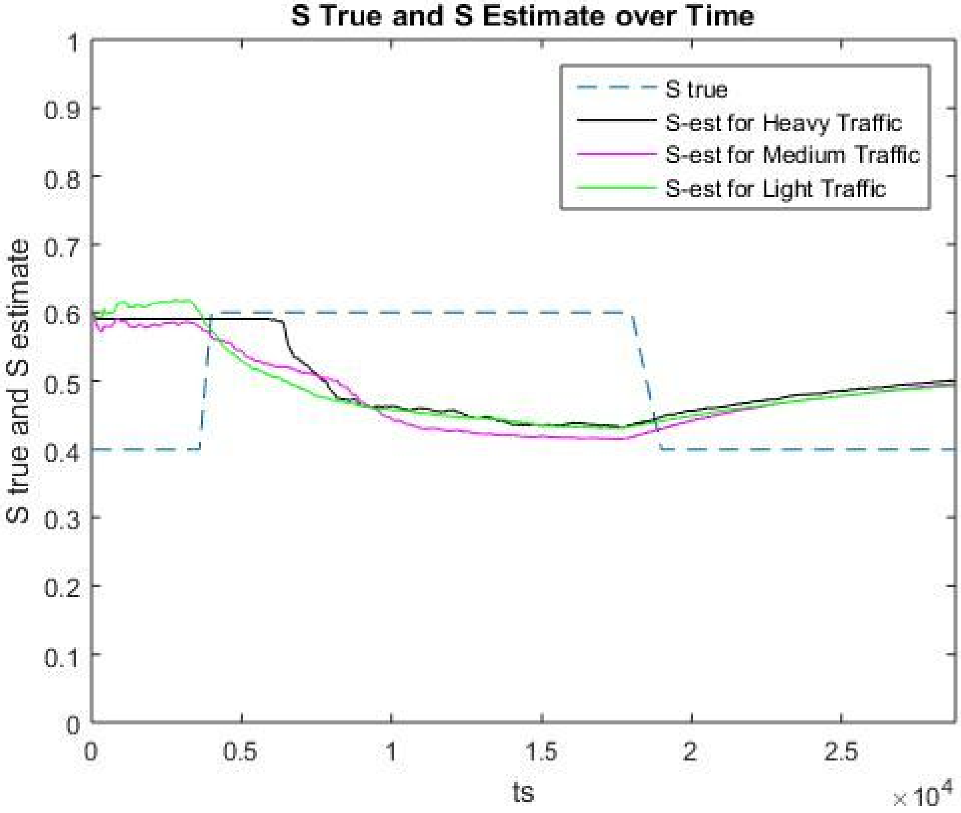


Figure: S True and S Estimate - Sample Engine

5. Description of Simulation Engine and FEL Implementation

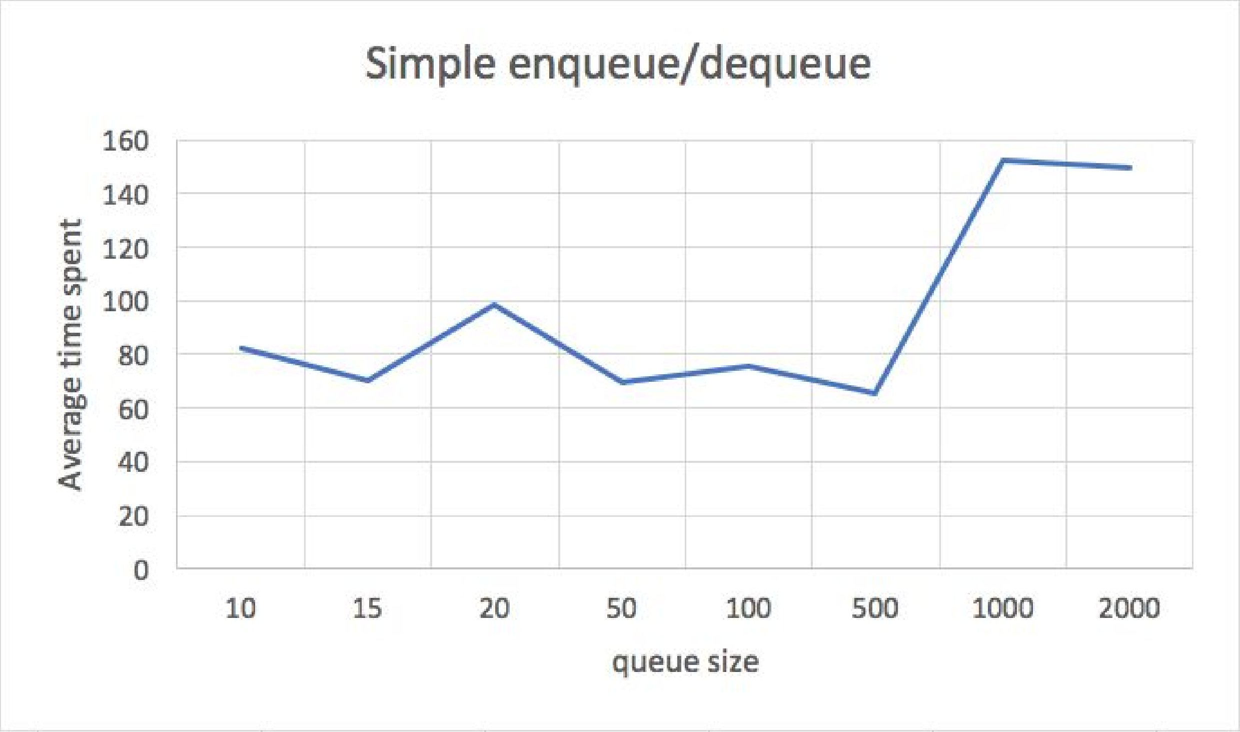
First, each event was structured as a node to implement Future Event List(FEL). The node contains three things: 1. Time stamp (priority). 2. Application data, and 3. A pointer to the next node. If the node is the last element of the linked list, then the pointer will be set as NULL.

The simulation application should initialize the main array and it further initialize an array with a length of 2 by pointing the subarray to the desired element of the main array. The Schedule function then put events into the bucket: i (bucket number) = time_stamp(ts)/bucket_width(width)%bucket_list_length(nbucket). During dequeuing, the function first finds the least priority timestamp bucket and dequeue the events in the bucket until the bucket top is reached. After that, the function goes on and search for the smallest time stamp. Depending on the queue size and the set threshold, the program will decide if the bucket list needs to be doubled or halved. During the event, the bucket list will first be copied. After the list length is change, all bucket will first be set to NULL and the events will be re-scheduled (put) into the calendar queue using Schedule function. In the process, a new_bucket_width will be calculated by sampling the events and calculate how far they are separated. The program will repeatedly dequeue the calendar until there's no event in the queue or the application time reaches end time.

6. Testing Procedure of Simulation Engine

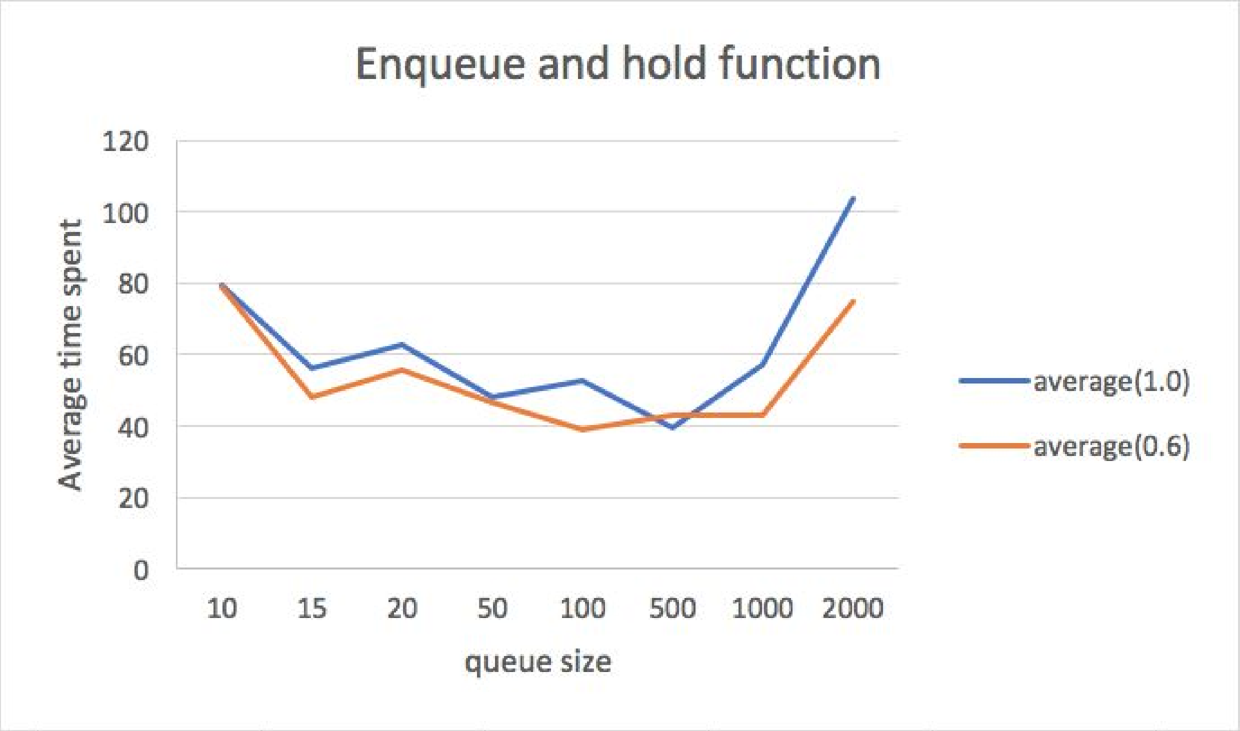
Two test was performed according to the Brown paper. One is simply enqueue and dequeue operation and the second one is a hold function, where the function insert an event at a random time in the future based on the timestamp that it was dequeuing an event.

The result of the simple dequeue/enqueue was shown in the following figure:



In the graph, although the relationship seems to be linear, it is actually more like a ladder like relationship, where the main factor that affects the time cost to do the operation is the resize() function. Since it copies dequeue and enqueue, it is a really expansive function to perform. After the size of the list is stabilized, the algorithm can reach O(1).

The result of 0.6 and 1.0 hold function is shown in the following figure:



The result is similar to Brown's paper. We can also notice that the average time spent on the hold operation is less expensive compare to simple queue/dequeue operation. The main reason is that the hold function keeps the queue size the same so the program will not go through resize twice; therefore, the cost of the operation is relatively smaller.

7. Results of Test Runs Integrated Simulation Application and Simulation Engine

The code was run with the simulation engine from the problem statement. The same metrics as previously mentioned, were used to verify that the code was working correctly. A sample result of the final numbers of incoming calls, summing to 100% are reproduced below.

Table 3: Incoming Call Metrics with Simulation

	Percent No Waiting	**Percent Hung Up**	**Percent in Queue**	**Total**
Light (1.5)	99.589714	0.110077	0.300210	100%

7.3 Simulation Engine

As shown above, the simulation is working appropriately while running on testing application. After the initialization, the program output the event list and print out the priority of the dequeued event. The timestamp and the eventlist was checked. Furthermore, the resize function is activated at the right time and the new width of the bucket was calculated accordingly.

8. Results of Experiments of the Simulation Application and Engine

Since the incoming calls are modeled as a poisson distribution, the rate of incoming calls can be set by a rate parameter, which is equal to 1/A, which is the average time between calls. The application was run for light, medium and heavy traffic. The table below shows results from the simulation application with the simulation engine. The light traffic condition used a rate parameter of 1.5. Because of differences in the engine, and the capability of it to run with a higher rate parameter, the medium and heavy traffic flows were chosen as rate parameters of 5 and 10 respectively.

The following table and figure show the metrics for incoming calls. As both the figure and graph indicate, the fewest people had to wait in the queue for the light traffic conditions, and most people had to wait in the queue for heavier flow conditions. The percentage of people that hung up are greatest in the heavy traffic condition because enough people had a chance to enter the queue to begin with. The number of people that hang up is based on a random number generator and they have a 25% chance of hanging up, which is in line with my results for the heavy traffic condition.

Table: Incoming Call Metrics

	No Wait	Hung Up	Entered Queue	Total
Light (1.5)	99.784378%	0.053326%	0.162296%	100.0%
Medium (5)	83.197800%	4.396007%	12.406199%	100.0%
Heavy (10)	0.69501%	25.659401%	73.645576	100.0%

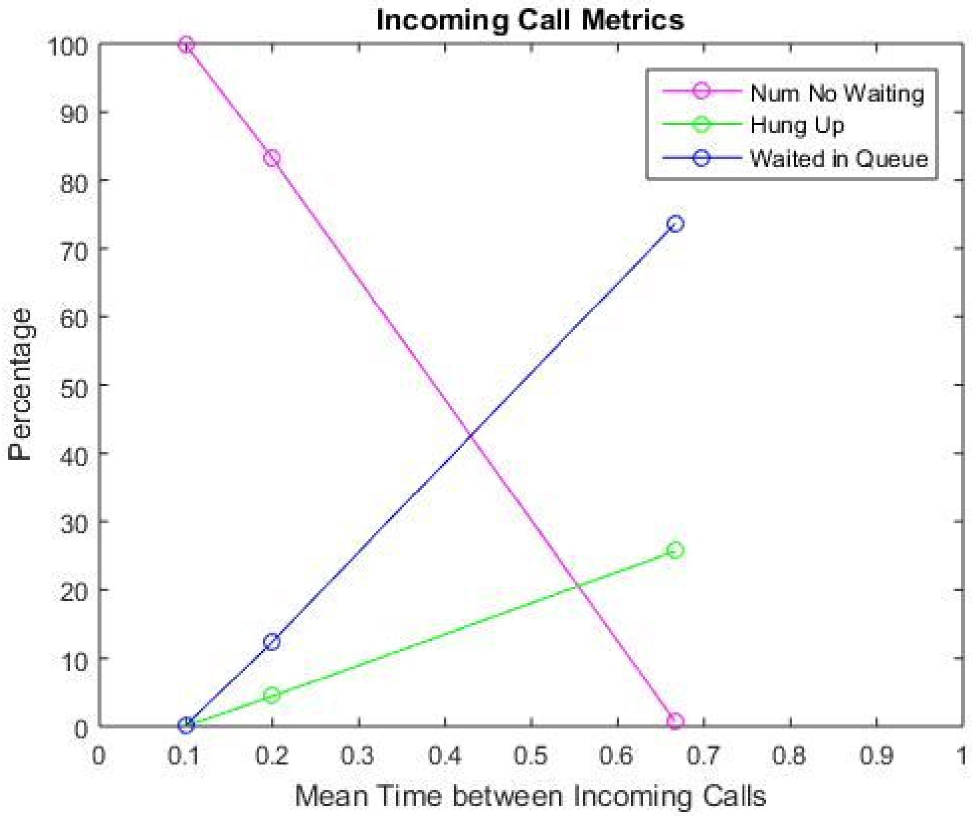


Table: Idle Service and Sales Agents Metrics

The average number of idle (not on the phone) service and sales agents over a 60 second period were computed. The number of idle agents would be the greatest in the light traffic scenario and the smallest in the heavy traffic scenario. This is expected, especially for the number of idle service agents, because they can only deal with incoming calls, and the rate parameter sets the frequency of incoming calls. The number of sales agents available also decreases with heavier traffic because they need to help deal with incoming calls, while also dealing with outgoing calls.

	Idle Service Agents	**Idle Sales Agents**
Light (1.5)	24.47	32.3
Medium (5)	8.5	4.4
Heavy (10)	2.15	0.9

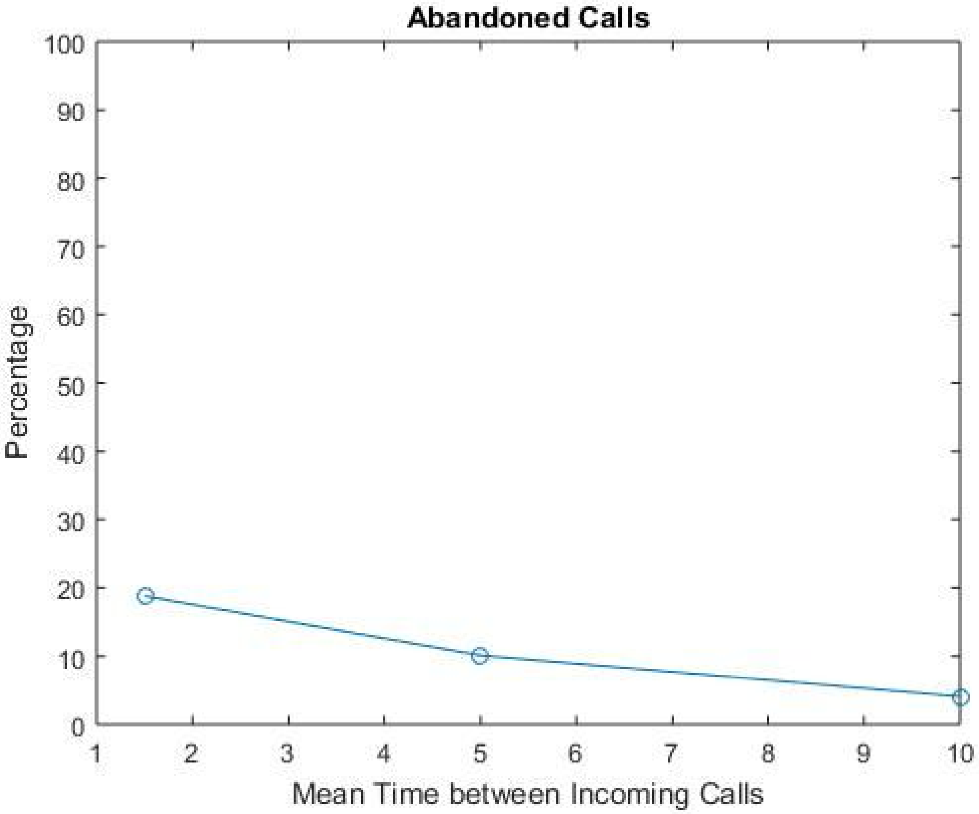
I was unable to compute the average wait time with the simulation application. Theoretically, I would mark the current time that an incoming caller entered the queue and the current time that a caller exited the queue, subtract the two values and then sum and average over the total number people that entered the queue.

Similar metrics were computed for the outgoing calls. Sample results for the total number of abandoned calls, the average number of abandoned calls per hour, the average number of successful calls per hour, and the percentage of successful calls that were abandoned are shown in the table below.

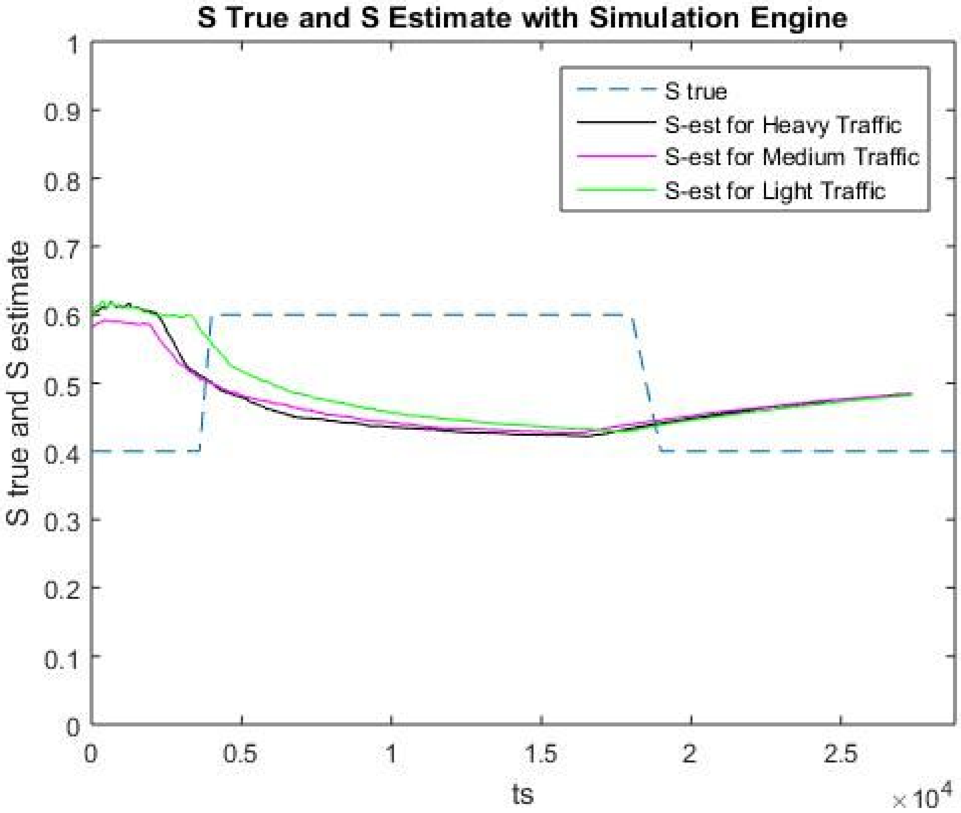
Table: Outgoing Call Metrics

	Total Abandoned	**Average Abandoned per hour**	**Average Successful Per Hour**	**Percent Abandoned**
Light (1.5)	1131	141.375	2667.625	18.869143%
Medium (5)	109	13.625	138.125	10.137614%
Heavy (10)	186	23.25	94.875	4.080645%

Based on the table, the greatest number of outgoing calls occurred during light traffic. This may be because there are fewer incoming calls, and therefore, more sales agents available at a given time. During heavy traffic, there is the lowest number of average successful calls per hour because the sales agents will be busy helping the service agents out with incoming calls, thus decreasing the number of successful calls per hour. The percent of successful calls that are abandoned are lowest for heavy incoming call traffic. This may be because there are much fewer calls being attempted to begin with, so there are fewer calls that are abandoned. The percentage of abandoned calls as a function of mean time between incoming calls is plotted below.



In the figure below, the values of S True and S Estimate were plotted with the simulation engine. The shape of the graph is similar to that produced by the sample engine, except this one is a bit smoother. Like the previous graph, it shows that S estimate lags behind S true, but does show signs of eventually adapting to the new value. If the simulation were allowed to run for longer period of time, perhaps the value of S estimate would eventually converge to be approximately equivalent to that of S true, assuming the value of S true remains constant.


S True and S Estimate with Simulation Engine
S true
S-est for Heavy Traffic
S-est for Medium Traffic
S-est for Light Traffic
S true and S estimate
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
ts
×10^4