Analysis of the Repair Network of Toner It Down!

Abby Macaluso, Brittany Stenekes, Isabel Coto, Kritika Sekar, William Xiao
ORIE 4580 Final Project

Abstract

Toner It Down! Inc is a leading developer and manufacturer of high-tech document management systems. However, as of late their customer support and repair system has been rather lacking. In this report, we aim to analyze their current repair network to determine the most cost-efficient way to bring the level of repair service in line or better than that of their competitors. We first discuss the mathematical/simulation model we used to model the system, and justify its correctness. We then run many different trials, changing various parameters such as the number of mechanics and vans hired, to see how each changes the repair service level. Finally, we analyze the results and stipulate that hiring 3 mechanics and 8 vans is indeed the lowest-cost option for Toner It Down! to bring its repair service to its desired levels.

I. Executive Summary

We have completed a full analysis in order to find the appropriate number of mechanics and vans to beat the competitor's initial response time and copier replacement time while keeping costs at a minimum. We used previous data from the Toner It Down network to form a model and improve customer service.

We chose the least number of mechanics and vans that resulted in an average response time less than the competitor's one hour and an average copier replacement time less than the competitor's three hours. This resulted in eight mechanics and three vans in order to ensure that costs were kept as low as possible.

Eight mechanics and three vans produced an average response time of 0.602 hours, which is 36.12 minutes compared to the competitor's one hour. It also takes 2.292 hours to replace a copier compared to the competitor's three hours. With this new analysis, Toner It Down! has a response time that is 39.8% faster than that of the competitors. It has a copier replacement time that is 23.6% faster than competitors. The number of vans and mechanics keep costs low at $1,420,000 per year to operate these resources. The cost for mechanics is $1,120,000 and $300,000 for vans.

In order to find these results we used past data to form a model and then verify and draw conclusions from it. From our past data we were able to estimate the percentage of request arrivals from each business center. We found the different arrival rates for each three hour period across the 24 hour day and the distribution of the initial diagnosis time. We then used these percentages and distributions as parameters in our model.

We implemented one optimization tool in our model. We kept the mechanic at the location of the business center it just serviced until it was assigned to another business center. This decreases the travel time a mechanic takes between servicing business centers. Instead of going back to the dispatch center after every call to a business center, the mechanic can now go straight to the next job. Toner It Down would have to implement this tactic in order to achieve these results.

II. Introduction

Toner It Down! Inc (TIDInc) develops and manufactures high-tech document management systems which includes products such as printers, scanners and fax machines. TIDInc has three major departments: production-sales, consulting, and repair. The repair division supports the equipment that has already been sold and it is the department that we would like to focus our study on. In the Syracuse area, TIDInc's repair network's performance is lacking and as a result, they are serving their customers much slower than their competitors. Unless they are able to improve their performance and their repair network in the Syracuse area, TIDInc will lose their customer base to their competitors.

Our study focuses on the local Syracuse repair network and the number of resources required to fulfill customer requests on time. For this initial study, we are focusing on one specific model of copiers that is currently used by

10 business centers in Syracuse. We are interested in two specific metrics at this time: the initial response time and the delivery time for replaced copiers. The initial response time the amount of time it takes for a mechanic to reach the customer location (one of the 10 business centers) after the customer called. The delivery time for replaced copiers is the amount of time it takes for a functional copier to be delivered and installed for the customer, if the copier is broken, after the customer calls. We want to reduce the time for both of these metrics in order to optimize the network and achieve a faster overall service time.

Currently, the competitors in Syracuse have an average initial response time of one hour and the delivery time for replaced copiers is three hours. Our goal in this study is to estimate the number of mechanics with cars and vans needed to achieve an economical repair network that will perform on par with the competitor's network, so that TIDInc does not lose market share in Syracuse.

III. The Simulation Model

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In feugiat felis eget mi tristique sagittis. Praesent blandit, dui eget tempus volutpat, erat diam luctus felis, ac vehicula quam leo non ipsum. Phasellus in sem nec neque congue congue. Fusce nec diam vitae quam ullamcorper elementum. Nullam id commodo enim. Vestibulum nulla eros, viverra et orci at, euismod volutpat ante. Pellentesque justo dolor, posuere sed aliquet vel, facilisis ac risus. Vestibulum lobortis ullamcorper ultrices. Curabitur in interdum odio, id dictum elit.

I. Our Approach

We created a map including the ten business centers and single dispatch center. We recognize that there is an initial response time: the time between when one of the customers calls to report a printer malfunction and the time that the mechanic reaches this customer. We looked at the data provided for the time of day customer requests are received and were able to estimate how often requests are in fact received. We also looked at the record of request locations and were able to determine how likely it is for the request to be from each of the ten business centers. We define our specific findings in the Data Analysis section below.

By measuring the distance from the dispatch center to each of the ten business centers we can determine how long it takes for a van to travel between locations. In our simulation, we utilize the fact that the vans travel at 60 kilometers per hour to determine the time it takes for a van to travel between the dispatch center and business centers.

We also recognize that there constraints on the business operations. Since a van can only carry a single copier at a time, in our simulation we have ensured that after reaching a customer to swap a broken copier, the van must return to the dispatch center before responding to any other customers because they need to pick up another copier. Therefore, we account for the time it takes for the van to drive in two directions: to the customer and back to the dispatch center. In addition, we recognize that swapping malfunctioned copiers with new copiers takes time–both at the business centers and at the dispatch centers. At the dispatch center, since it takes time for the van operator to swap the broken copier they picked up for a new copier, we have incorporated a delay into our simulation, such that the van does not depart the dispatch center again until time has passed. Likewise, at the business center, once a van has arrived at a business center, we include another delay time and only allow the van to leave once the replacement service is completed. Given that swapping copiers at the dispatch center takes as little as 10 minutes, as much as 25 minutes, but usually about 15 minutes, we are able to model the service time at the dispatch centers to reflect the measurement. Additionally, since swapping copiers at a business center takes as little as 20 minutes, as much 60 minutes, but usually about 30 minutes, we are able to model the service time at the business centers according to these time values.

We also know that there is an ample supply of spare copiers at the dispatch center. Therefore, our simulation assumes that whenever a van needs to pick up a functioning copier from the dispatch center, there is always a copier available. And since every business center uses the same type of copier, our model assumes that once a van arrives at a business center, the van operator performs the same copier swap procedure they would perform at any other business center.

Taking these characteristics of the business center locations, van speed and customer call frequencies into account, and business constraints, we can simulate a day of business operations by applying these characteristics to our simulation runs.

II. Derivation of Input Distributions

Before we could start creating our model, we first had to analyze the past data and see if we could draw mathematical insights from the past data, so we could more accurately create a simulation that would reflect the real-world effects of increasing the number of mechanics and vans.

In order to do this, we took the sample data we were given from the past few weeks of operating the business. In this data, we were given the time of day for the call, initial diagnosis times, origin (business center), onsite repair times, the number of offsite repairs, etc. From this, we needed to glean some key insights that were not explicitly given in order to more properly model this system, including:

- How often do we receive requests? Do certain times of day have more requests than others?
- What proportion of requests come from each business center?
- How should we model the amount of time it takes to travel from one location to another?
- How should we model the amount of time it takes to diagnose a system?
- How should we model the amount of time taken for an offsite repair and swap?

Before we began processing the data, we had to clean it up and remove outliers that made no sense in order to preserve the accuracy of our data and remove errant noise.

Some of these values were easier to calculate than others. For example, modeling the proportion of requests that came from each data center was quite simple - we simply found the number of requests from each data center divided by the total number of requests received, and used this empirical proportion as a good estimate for the actual proportion that come from each center. Calculating the amount of time to travel from one location to another was also quite simple. We were given/assumed for simplicity that all vans move at 60kph, and we also had a map that gave us the Manhattan distance between each of the business centers and the dispatch center. From there, we calculated the estimated amount of time using simple division.

Others were more difficult to compute and model. For example, the rate at which we receive requests was not constant, and thus could not be modeled simply. We had to figure out how many, on average, happened per hour, and then used this to more accurately come up with a formula to model how many requests arrived based on the time of day. Perhaps unsurprisingly, more requests happened during the day/working hours than at weird hours of the night/early hours of the morning.

Similarly difficult was figuring out how to model the amount of time it takes to diagnose a system, or to complete an offsite repair. For the former, it was not enough to generalize a formula for all business centers, because grouping all the diagnosis times together did not appear to resemble any distribution/pattern we had seen before. We had to split it up by business center, and then do analysis from there - noticing that different groups business centers shared similar characteristics in diagnosis times, and then recombining and analyzing from there to figure out how to model said diagnosis times. Calculating offsite repair times was much the same - we had to take the data together, and once we realized that it was similar to distributions/patterns we had seen before, we figured out what specific numbers we needed to model the pattern mathematically so we could simulate it.

A more technical and detailed analysis of the derivations of the statistical distributions we used to model the system can be found later in the appendix.

III. Verifying the Model

In order to verify our model, we need to make sure each subsection of the model makes sense in context. Before we do a deep dive into each individual section, we will first go over the original results that the simulation outputs. We found that, after running several initial simulations and comparing the results, that hiring 8 mechanics and 3 vans was indeed the optimal number of repair staff to hire in order to bring down the performance metrics to their stated levels (the bulk of which detail will be discussed in the modeling results). These results initially did in fact make sense, as we should need much more mechanics than we do vans. The mechanics are always on call, and driving around to various business centers to see what the issue is with the copiers. They then have to diagnose the issue, fix it if possible, and return home or go to the next business center to diagnose that issue. The vans are needed much more infrequently in comparison, only being called when the mechanic could not fix the problem onsite and needed the van to bring a replacement copier as backup to install in the customer location and bring the defective one back to the service center. In general, it would make sense that the amount of mechanics we need is much greater than the number of vans - we need vans less often than we do mechanics.

The next thing we found that we needed to verify was that the distributions for interarrival times, etc were being generated correctly. We verified this by creating lists that stored all the times of note, and then plotted those values in histograms to make sure they matched the distributions they were generated from, and indeed they did. As some examples, for the onsite diagnosis times for the business centers, we made histograms to capture the times, and plotted them against the pdf of the underlying distribution they should have been generated from.

Mean 0.3962 hours $\approx$ 23 minutes for onsite repair time for business centers 2, 3, and 9.

Mean: 0.26854 hours $\approx$ 16.1 minutes (mean 16 diagnosis) for other business centers. 0.3682661317472747 (mean 22 diagnosis) The histograms below verify that the distribution is being generated correctly.

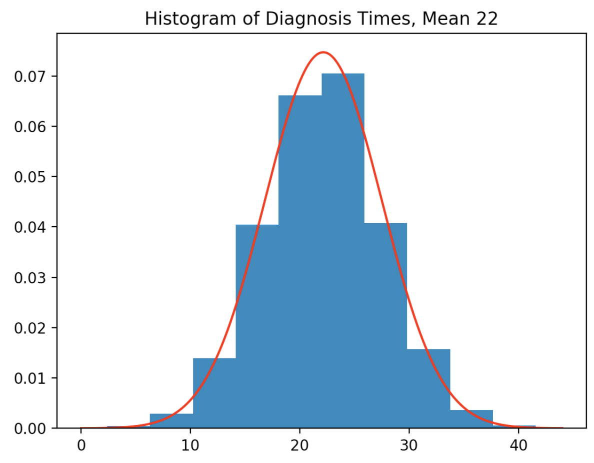


Figure 1: *Histogram of initial response times vs pdf, mean 22.*

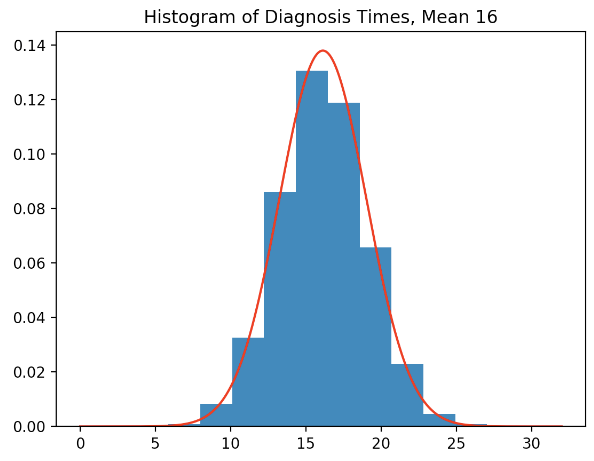


Figure 2: *Histogram of initial response times vs pdf, mean 16.*

Similar analysis for the onsite repair times yields the following:

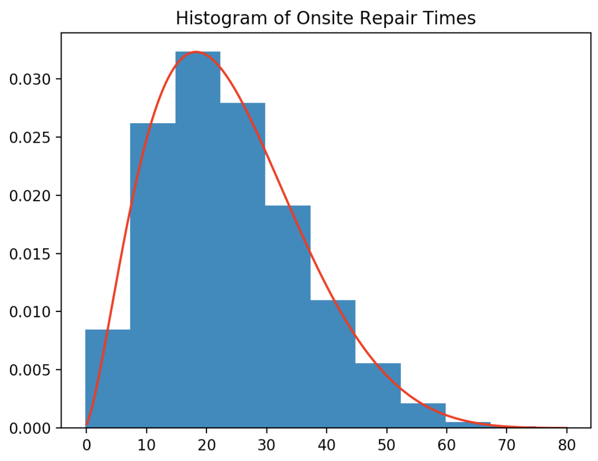


Figure 3: *Histogram of onsite repair times vs pdf.*

The second part of our model verification includes running a simplified simulation of the van and mechanic behaviour over 24 hours at TID inc and comparing the simulation results to known calculated values from queuing theory. To apply queuing theory, we will suppose arrivals of requests to TID inc and to the mechanics and vans are independent Poisson Processes. We will also assume exponentially distributed service times. The following simplifications were made to the model in order to verify it:

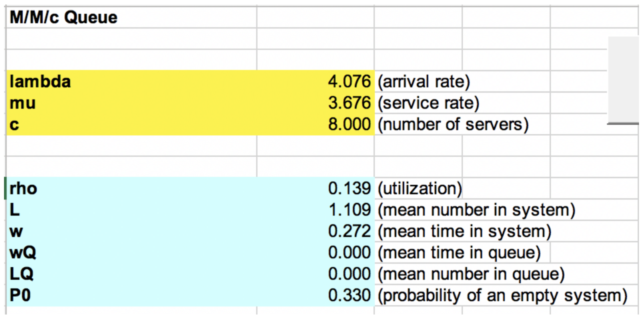
1. Suppose requests are equally likely to come from a distribution center
2. Suppose the rate of arrivals of requests is a Poisson Process with constant rate 4.07625/hr. (This rate is the average rate of the non-homogeneous poisson process that the real model uses)
3. Suppose the time it takes for a mechanic to travel between all business center and distribution center locations takes 0.62173 hours on average (which is the average of the all the travel times for the origin-destination pairs). The mechanic has an average diagnosis time of 0.27205 hours . So we let the mechanic's travel time be exponential with mean 0.62173 and the diagnosis time to be exponential with mean 0.27205
4. Suppose the time it takes for a van to travel to and from business centers takes 0.606047 hours (which is the average of the all the travel times going from the dispatch center to a business center). We use 0.611 hours for the average onsite swap time for a van.

We use 0.833 hours for the average offsite offloading time for the van. Suppose the van's fix time = travel + swap time is exponentially distributed with mean 0.606047 + 0.611 = 1.217 hours. And Suppose the van's refresh time = travel back + offload is exponentially distributed with mean 0.606047 + 0.833 = 1.493 hours.

5. Suppose the time it takes for a mechanic to perform an onsite repair is exponential with mean 0.260 hours.

6. Suppose there are 8 mechanics and 3 vans.

We get the following theoretical results when making these simplifying assumptions:

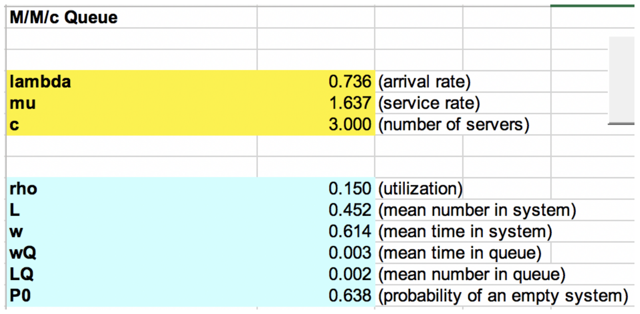


M/M/c Queue		
lambda	4.076	(arrival rate)
mu	3.676	(service rate)
c	8.000	(number of servers)
rho	0.139	(utilization)
L	1.109	(mean number in system)
w	0.272	(mean time in system)
wQ	0.000	(mean time in queue)
LQ	0.000	(mean number in queue)
P0	0.330	(probability of an empty system)

Figure 4: *Theoretical result for mechanic diagnosis times.*

The time it takes for the request to receive a mechanic and finish the diagnosis = initial response time + mean time in mechanic diagnosis = 0.62173 + 0.272 = 0.894 hours.

For the onsite van repair times:



M/M/c Queue		
lambda	0.736	(arrival rate)
mu	1.637	(service rate)
c	3.000	(number of servers)
rho	0.150	(utilization)
L	0.452	(mean number in system)
w	0.614	(mean time in system)
wQ	0.003	(mean time in queue)
LQ	0.002	(mean number in queue)
P0	0.638	(probability of an empty system)

Figure 5: *Theoretical result for onsite van repair times.*

The time it takes for the request to receive a mechanic and finish the diagnosis, call a van, the van to travel to the business center and for the van to replace the printer = initial response time + diagnosis time + van travel + time van spends on site = 0.894 + 0.606047 + 0.614 = 2.11 hours.

Then, we then run the simulation and compare our answers from the simulation to the theoretical results and get the following:

Simulation Configuration: 95% confidence intervals, 250 replications

- Average initial response time = 0.6511866023525092, with 95% CI = (0.64718660235251, 0.65518660235251)
- Average time between request and diagnosis finished = 0.92414109133761, with 95% CI = (0.91814109133761, 0.93014109133761)
- Average time to replacement = 2.0659709999418365, with 95% CI = (2.02897099994184, 2.10297099994184)

The simulated results correspond closely to the theoretical expectations for the simplified version of the model. Now that we have confidence in our model, we will change the distributions of events to closely resemble the data TID Inc. provided us with.

IV. Running The Simulation

In running the simulation, we decided to make each trial last 24 hours, as we are trying to model the number of mechanics and vans we need to hire to make sure each day is dealt with - the data we were provided was on a per-day basis. Trying to extend this to per week or per month would be counterproductive.

Also, we decided to run the simulation for 250 replications each time to try to minimize the amount of noise while still providing relatively accurate confidence intervals - doing too many more would cause the simulations to run extremely slowly, while doing less would allow for the possibility of outliers to severely skew the data.

We made the number of mechanics and the number of vans constants that we could easily change in our code in order to see how changing those numbers affected our times/quantities of interest.

V. Simulation Results

With our model now generated and justified, we can now use it to help answer the questions that Toner It Down! is interested in. Our main goal was to find out what number of mechanics and vans to hire such that:

- the initial response time is less than one hour.
- the customer has a working copier within three hours, on average, of the customer call.

is achieved for the minimum possible cost. These numbers came from comparing to the competitors in the area - these are approximately how well the competitors are doing, and we are aiming to do as well, if not better.

With the goal in mind, we could then begin to figure out the optimal number. Well, there are two variables/parameters we need to find optimal values for: the number of mechanics, and the number of vans. Since it is somewhat more difficult to find the optimal value for two variables at once, we opted to optimize one variable first, holding the other constant, then using that to optimize the other one. This gave us a good starting point for what number mechanics and vans to hire, and we could perturb the values as necessary from there to show that it was indeed the optimal number to hire.

First, we decided to optimize the number of mechanics. In this experiment, we ran the simulation 250 times (each simulation representing a day), keeping the number of vans fixed at 1. We tried varying the number of mechanics between 1 and 10 for this simulation, and got the following box and whisker plot for the initial response times for each number of mechanics:

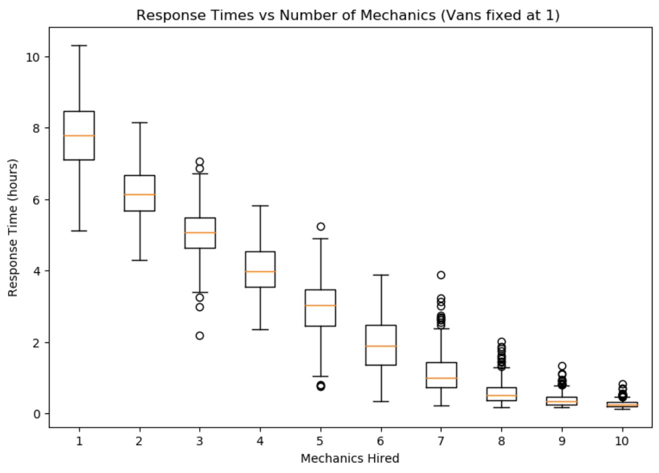


Figure 6: *Boxplot of response times vs. mechanics hired.*

In this diagram, the orange line represents the mean of the data for that number of mechanics, while the outside circles represent outliers. One can observe that the the response time seems to be negatively correlated with the number of mechanics hired, as expected. The more mechanics we hire, the lower the response time gets (up to a certain point). The amount of benefit gained from hiring more mechanics decreases once we hire too many of them (around 7 mechanics or so is when it starts to flatten out). If only one mechanic is hired, then the response time hovers around 8 hours - a far cry from the 1 hour or less time goal we aim to achieve. We first notice the mean of the response times decrease to around the 1 hour mark around 7 or so mechanics hired. To account for possible fluctuations in the value of the response time when we add more vans, we decided for now to use 8 mechanics as the temporarily optimal number of mechanics to hire. With this number chosen, our next task was to find the optimal number of vans to hire in order to bring down the average time to replacement to under 3 hours.

As previously, we ran the simulation 250 times (each simulation representing a day), keeping the number of mechanics fixed at 8. We tried varying the number of vans between 1 and 4 for this simulation, and got the following box and whisker plot for the replacement times for each number of vans:

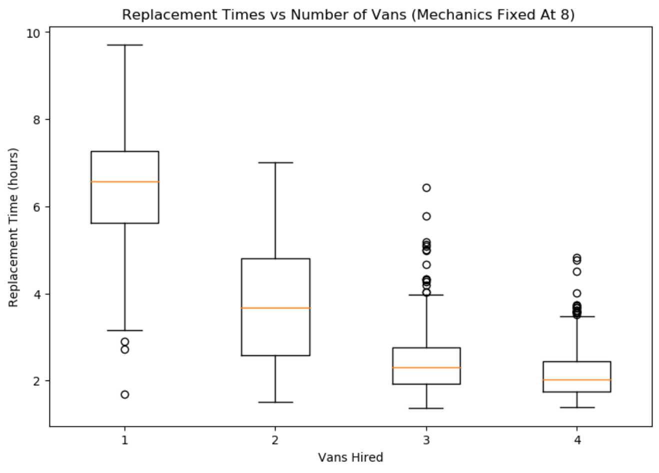


Figure 7: *Boxplot of replacement times vs. vans hired.*

Similarly to the previous diagram, the orange bar represents the mean replacement time for that number of vans hired. We can see, similarly to before, that the replacement time is negatively correlated with the number of vans hired - that is, the more vans we hire, the smaller the replacement time gets - this happens because there are more vans available, on average, to service the offsite repair requests and replace the copiers - clients spend less time waiting for one to be available in the first place. Most of the waiting time happens in the travel time to get to the center entirely. We can see that, assuming we hire 8 mechanics, if we hire 1 van, then the replacement time sits around 7 hours, much more than our 3 hour goal. The average replacement time also decreases exponentially - that is, the more vans hired, the smaller the benefit of hiring a new van becomes. The 3 hour goal is achieved when hiring 3 vans with the 8 mechanics, at a replacement time of around 2.7 hours.

So, our baseline number of staff to hire is currently 8 mechanics and 3 vans. When doing so, we get the following estimates for mean times, with associated 95% confidence intervals:

- Average response time: 0.5897298401769622, with CI (0.5487298401769621, 0.6307298401769622)
- Average time to replacement = 2.50434268300868, with 95% CI (2.40334268300868, 2.60534268300868)

These times of 0.5897, 2.5043 are well within our bounds of 1 and 3 hours. We now see if we can decrease either of them and still be within our time goal.

If we decrease the number of mechanics by 1 (so 7 mechanics), and run with 3 vans, we get the following numbers:

- Average response time = 1.0957024883727158, with CI (1.0267024883727158, 1.1647024883727157)
- Average time to replacement = 3.3574510686483556, with 95% CI (3.202451068648356 3.5124510686483554)

We can see that this is non-optimal - the average response time is slightly larger than that of the competitors, while the average time to replacement is noticeably larger.

We then try and decrease the number of vans by 1 (so 2 vans, 8 mechanics) to see if that would cause similar results in the average times. In doing so, we get:

- Average response time = 0.6117634671737309, with 95% CI (0.5707634671737308, 0.6527634671737309)
- Average time to replacement = 3.8029383144588, with 95% CI (3.6419383144588, 3.9639383144588)

We can also see this is nonoptimal - while the average response time is still well in line, the average time to replacement is about 30% larger than our target. So, this will not do either.

Decreasing both by 1 (so 7 mechanics, 2 vans) yields:

- Average response time = 1.08764983794, with 95% CI (1.01164983794, 1.16364983794)
- Average time to replacement = 4.413527219398, with 95% CI (4.235527219397, 4.59152721939737)

As we expect, decreasing both by 1 leads to even worse times. So, we can safely assert that 8 mechanics and 3 vans is indeed the optimal number of each to hire. Increasing either would only decrease the times further. While it would be better for the customer, it also increases costs significantly for TID! Inc.

So, with 8 mechanics and 3 vans, we can say that the cost of this optimal solution is Annual Costs = ($140,000 / mechanic * year) (8 mechanics) + ($100,000 / van * year) (3 vans) = $1,420,000 / year.

I. Sensitivity Analysis

With the original parameters for the functions in the code, the average response time we observed was 0.641402384 hours with a CI of (0.59740238,.068540238) and the average time for replacement was 2.5747399 hours with a CI of (2.4737399,2.6757399). We want to investigate the effects of changing the parameters of various functions on the estimated average response time and time to replacement. Doing so will measure how sensitive our model is to these parameters and fluctuations in real life. We begin this sensitive analysis by changing the parameters for Diagnosis Time, Repair Time on Site and Arrival Rates because these values were directly from previous data and are subject to more fluctuations whereas for other values, we were provided with more concrete information that was derived from longer periods of time. For the Diagnosis Time and the Repair Time on Site functions, we observe the change in the average response time and the average replacement time when the parameters are changed by $\pm$ 10% of the initial parameter value. For the Arrival Rates, because we have confidence intervals, we change the parameters to the lower and upper bound of the confidence interval in order to perform the sensitivity analysis. The results are highlighted in the following two tables, which can be found in the appendix. Table 9 displays the results for the lower bound parameter changes and Table 10 displays the results for the upper bound parameter changes.

Initial Parameters:

- Diagnosis Time For Business Centers with Mean 22: $N(22.15, 5.34)$
- Diagnosis Time For Business Centers with Mean 16: $N(16.13, 2.89)$
- Repair Time on Site: $Beta(2.6158, 7.4606, -.4622, 93.475)$
- Arrival Rates: [0.972, 3.233, 5.239, 6.311, 6.422, 5.617, 3.494, 1.322]

We are given the formula that:

Completion time for on-site repairs = initial response time + on-site diagnosis time + on-site repair time.

Thus, we know that the diagnosis time should not affect the response time significantly. It also shouldn't affect the replacement time because that is what the vans are for. As a result, perturbing parameters in the diagnosis time function should not have a tangible effect on the average response time or the average replacement time. From the tables above where we change each parameter individually for this function, we are able to see that the average times do not change very much and neither do the confidence intervals in relation to the initial values that we found.

Similarly, we perturbed the parameters for the average time for on-site repair and also noticed that the neither the average response time nor the average replacement time nor their corresponding confidence intervals change significantly They are very much in-line with our initial findings. This would make sense because only if the on-site repair time parameters increased by a very significant amount would we see the effect if maintain the same number of mechanics because it would take the mechanics longer to get to the next client.

Finally, we experimented with changing the arrival rates to their lower and upper bound of their respective confidence intervals and seeing their effect on the average response time and the average replacement time. As the table above highlights, these values did not change significantly either which shows that our model can account for natural fluctuations throughout the day. Therefore, we can be 95% confident that fluctuations in arrival rates do not significantly affect the average response time or the average replacement time.

Therefore, through the sensitivity analysis, we are able to deduce that the fluctuations in parameters do not cause the average response time or the average replacement time to differ drastically to the point where 3 mechanics and 8 vans cannot keep up with the demand.

VI. Conclusion

In order to beat the competitor's initial response time and time to provide a copier while keeping costs low, Toner It Down should use eight mechanics and three vans. With these numbers Toner It Down will have an average initial response time of 0.602 hours and an average copier replacement time of 2.292 hours. After completing our model, we adjusted the number of mechanics and vans in order to beat the competition in service time while keeping the number of mechanics and vans as low as possible. We recommend that Toner It Down use eight mechanics and three vans while implementing a new policy for their mechanics. After each mechanic is done with a job at a business center, they should remain at that business center until they are assigned to a new job. This decreases the amount of time that the mechanic takes traveling between jobs and saves the company money.

VII. Appendix

I. Technical Analysis of the Input Distributions

In order to properly model the system, we needed to use the data we were given and be able to extract key insights from it, such as how to model the time it takes to diagnose a system, how long it takes to reach a customer, etc. Once we had all the mathematical assumptions needed to properly model the system, then we could start building our system and be able to run simulations.

The first thing that needed to get done was to observe the data and clean it. As with all data collection, there is a chance that some of the values collected are garbage/meaningless/noisy. In the case of Toner It Down!, this meant the data points that were outside the proper range for time of day. The data was collected for each day, i.e. in a span of 24 hours. Some of the data points were outside of this range, and occurred at times of day greater than 24. This is clearly nonsensical, so we removed these data points from our considerations. We found that there were 9 outliers in the data set, with the corresponding times:

- 24.00353465
- 24.00263736
- 24.02713942
- 24.00457209
- 24.01377616
- 24.21077354
- 24.31245061
- 24.10108313
- 24.02048333

We simply filtered out those rows from our data before doing further analysis.

I.1 Initial Diagnosis Time

From there, our next task was to find how to model the initial diagnosis time for each repair call, that is, the number of minutes between when a mechanic arrives at the customer location and when he completes the initial diagnosis. Our first approach was to simply plot a histogram of the initial diagnosis times, normalized for density, to see if it looked like a particular statistical distribution. We ended up with the following plot:

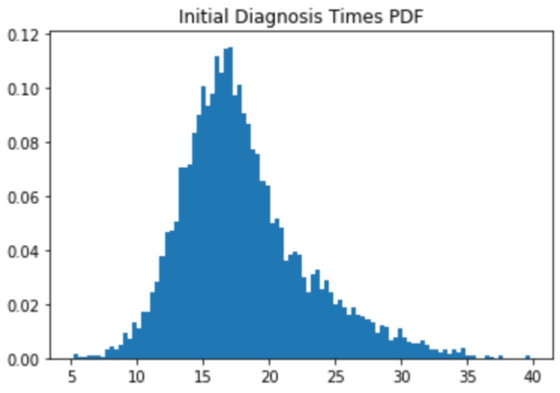


Figure 8: *Normalized histogram of the diagnosis times.*

This did not look like any distribution we had seen before. It seemed like a mix of the normal, beta, and triangular distributions, without having a heavy skew towards one or the other. At this point, we realized we had to analyze further and see if we could break down these diagnosis times into smaller categories to see if those could be fitted easily.

Our approach was to split up the data by business centers separately (one for each business center). When we did so, and found the corresponding mean and standard deviation for each one, we came up with the following:

Center	Mean	Std
BC_1	15.974684	2.954169
BC_2	22.174312	5.248368
BC_3	22.009613	5.405268
BC_4	16.175101	2.851679
BC_5	16.200575	2.882792
BC_6	16.151506	2.847626
BC_7	16.011773	2.923965
BC_8	16.269802	2.875527
BC_9	22.236205	5.358455
BC_10	16.109307	2.919997

Table 1: *Initial Diagnosis Times per Business Center.*

Once we split it up as so, we could see that there was indeed some pattern to the data. Most notably, business centers 2, 3, and 9 all had similar means and standard deviations (22, 5.3), while all the others had a similar mean and standard deviation (16, 3). Therefore, we decided to group the data into two sections: the initial diagnosis times for business centers 2, 3, and 9, and then the diagnosis times of all the others. When we plotted histograms for the initial diagnosis times for each of these two sections, we got:

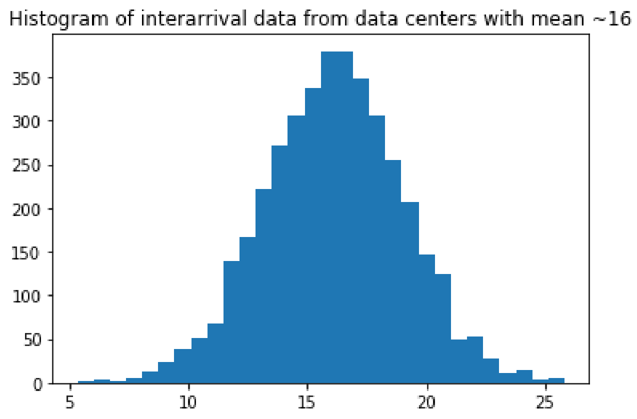


Figure 9: *Histogram of the diagnosis times with mean 16.*

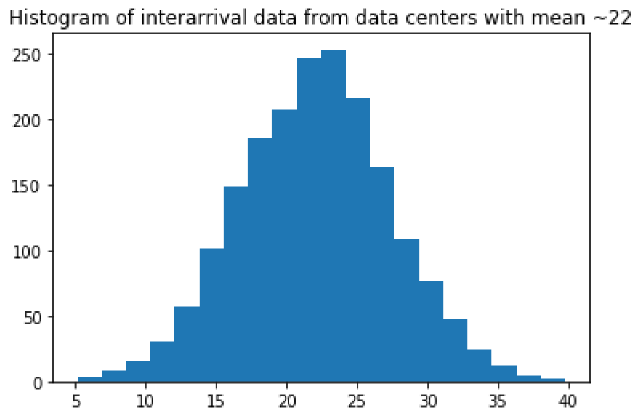


Figure 10: *Histogram of the diagnosis times with mean 22.*

Now, these distributions looked much more familiar. They appear to be normally distributed. So, our next approach was to fit the parameters to these distributions, and perform a K-S test to determine the goodness of the fit. After performing said fitting of parameters, we got the following:

- Diagnosis Times (mean 16): $N(\hat{\mu} = 16.13, \hat{\sigma} = 2.89)$
- Diagnosis Times (mean 22): $N(\hat{\mu} = 22.15, \hat{\sigma} = 5.34)$

To check that these distributions made sense, we ran the K-S test to compare and got p-values of approximately 0.85466 and 0.84999, respectively. These numbers are sufficiently high for us to proceed with the modeling assumption that the initial diagnosis time is distributed normally, with parameters based on the specific center.

I.2 Onsite Repair Times

The next task was to model the onsite repair times. We first created a histogram of the repair times to more easily visualize the distribution, as so:

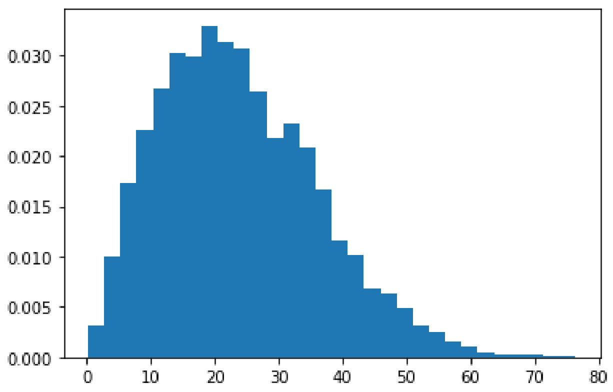


Figure 11: *Histogram of onsite repair times.*

From there, we tried many different distributions to see which one fit the data the best. We had a suspicion going in that the data was Beta distributed, as the histogram matched it the best. However, we still went and tested against various distributions like Normal, Rayleigh, Exponential, and Gamma. We then conducted K-S tests to see which distribution fit the data the best to get a numerical answer, rather than just a guesstimate. In the end, we observed that the repair times did indeed seem to be beta-distributed. So, from there, we fit the parameters to the Beta distribution, and performed the K-S test as well as generated a QQ-plot to test the goodness of the fit, and found that the fit was quite good. We ended up with the following distribution:

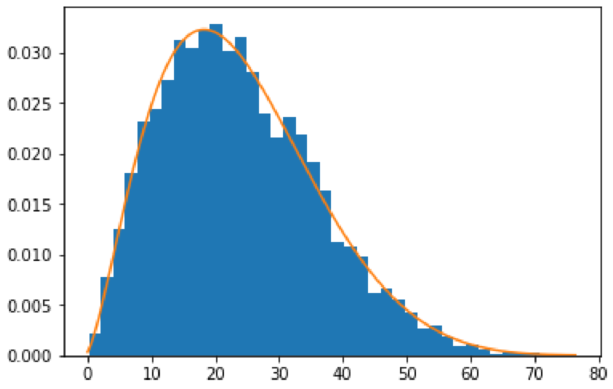


Figure 12: *Repair times, fitted to beta distribution.*

The resulting beta distribution had the following parameters:

- $a = 2.6158$
- $b = 7.4606$
- $loc = -0.4622$
- $scale = 93.4751$

The associated p-value of this fit was ≈ 0.69790, and the Q-Q plot looked as follows:

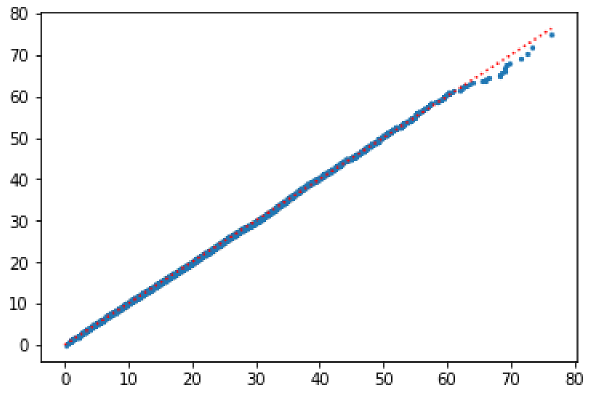


Figure 13: *Q-Q plot for fit of beta distribution to onsite repair times.*

Both of these metrics in parallel support our speculations that the onsite repair times are indeed beta distributed.

I.3 Arrival Rate

The next task was to model the arrival rate function. We decided to model these arrivals as a Poisson process, which was the most natural solution. When calculating the arrival rates, the first thing we did was to try to split up the arrivals by the day they occurred on. From there, we calculated the hour of arrival for each arrival of each day, and created a map that mapped the day to the hour numbers of the arrivals for that day. Then, we counted the number of arrivals for each hour of each day. We then aggregated the data from all days in order to find the arrivals that happened each hour. Finally, since we had the number of arrivals that happened each hour over the whole data collection time period, we finally then calculated the average number of arrivals. Doing so yielded the following rate function:

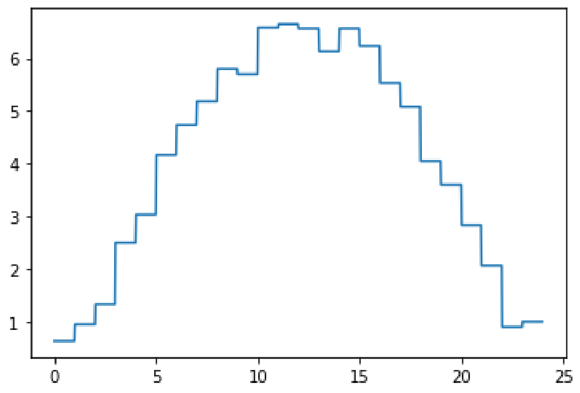


Figure 14: *Rate function of average number of arrivals per hour.*

We realized at this point that doing the rate function per hour was quite noisy - all the values seemed to vary more between hour than we would like. So, we decided to then try to compare the previously computed rate function to that if we did it every two-hour block, or every three-hour block in an attempt to reduce said noise and make the data more consistent (at the cost of some 'accuracy'). After comparing to both of the above, we decided that changing the rate function for three-hour blocks was the best option - it was much less noisy, and still had much of the same accuracy of the original data. The corresponding smoothed rate function looked as follows:

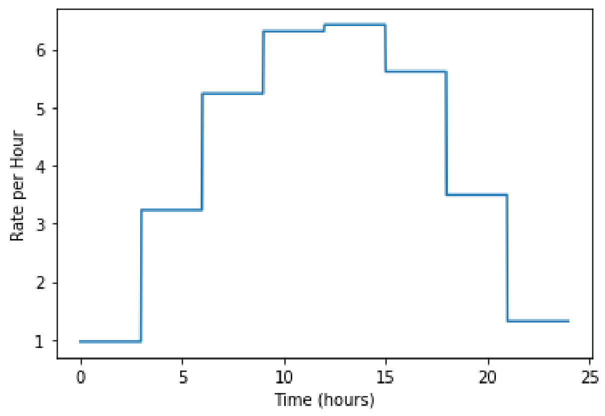


Figure 15: *Smoothed rate function of average arrivals / hour.*

Time (hour)	Arrival Rate
0..3	0.97222
3..6	3.23333
6..9	5.23889
9..12	6.31111
12..15	6.42222
15..18	5.61667
18..21	3.49444
21..24	1.3222

Table 2: *Rate table for arrivals.*

The entire rate function was generated using the following Python code:

```python
# Set up dictionary for separating
# arrivals per day
arrivals = {}

for i in range(1,61):
    arrivals[i] = []

# Calculate the hour of arrival for each data
# point and separate per days
for i in range(len(data)):
    dayNum = data["Day"][i]
    time = data["Time of day"][i]
    hourArrive = math.floor(time)
    arrivals[dayNum].append(hourArrive)

# Count arrivals in each our for each day
arrivalCountsDaily = {}

for key in arrivals.keys():
    count = Counter(arrivals[key])
    arrivalCountsDaily[key] = count

# Aggregate data from all days
# (arrivals within the hour)
arrHourCount = {}
for i in range(24):
    arrHourCount[i] = []

for day in arrivalCountsDaily.keys():
    for i in range(24):
        val = arrivalCountsDaily[day][i]
        arrHourCount[i].append(val)

# Calculate average number of arrivals
avgArrivals = {}

for hour in arrHourCount.keys():
    arr = np.array(arrHourCount[hour])
    avgArrivals[hour] = arr.mean()

# Plot the estimated rate function
x = np.linspace(0,24,1000, endpoint = False)
y = [avgArrivals[math.floor(val)] for val in x]

plt.plot(x,y)
plt.show()

# Every 3 hours
arrHourCount3 = {}
for i in range(0,24,3):
    arrHourCount3[i] = []

for day in arrivalCountsDaily.keys():
    for i in range(0,24,3):
        v = arrivalCountsDaily[day][i]
        v += arrivalCountsDaily[day][i+1]
        v += arrivalCountsDaily[day][i+2]
        arrHourCount3[i].append(v)

# Calculate average number of arrivals
avgArrivals3 = {}

for hour in arrHourCount3.keys():
    arr = np.array(arrHourCount3[hour])
    avgArrivals3[hour] = arr.mean()

# Plot the estimated rate function
x = np.linspace(0,24,1000, endpoint = False)
temp = []
for val in x:
    check = math.floor(val)
    if check < 3:
        temp.append(0)
    elif check >= 3 and check < 6:
        temp.append(3)
    elif check >= 6 and check < 9:
        temp.append(6)
    elif check >= 9 and check < 12:
        temp.append(9)
    elif check >= 12 and check < 15:
        temp.append(12)
    elif check >= 15 and check < 18:
        temp.append(15)
    elif check >= 18 and check < 21:
            temp.append(18)
        elif check >= 21 and check < 24:
            temp.append(21)
        else:
            print("ERROR")

    y = [avgArrivals3[val2]/3 for val2 in temp]

    print("Estimated rate function:")
    plt.xlabel("Time (hours)")
    plt.ylabel("Rate per Hour")
    plt.plot(x,y)
    plt.show()
```

I.4 Breakdown of Calls by Business Center

We then needed to find the proportion of calls that came from each business center. This was done by simply taking the data, and seeing how many came from each center and dividing that by the total number of requests. We then found 95% confidence intervals for each section and obtained the following results (rounded to 4 decimal places for clarity):

Center	Percentage	95% CI
BC_1	0.0394	(0.0344, 0.0443)
BC_2	0.0821	(0.0751, 0.0891)
BC_3	0.1089	(0.1009, 0.1168)
BC_4	0.1317	(0.123, 0.1403)
BC_5	0.1193	(0.111, 0.1275)
BC_6	0.0545	(0.0487, 0.0603)
BC_7	0.1276	(0.1191, 0.1361)
BC_8	0.0569	(0.051, 0.0628)
BC_9	0.1349	(0.1262, 0.1437)
BC_10	0.1448	(0.1358, 0.1538)

Table 3: *Percentage of Requests By Business Center.*

We could then split up our rate function to have the number of requests for each business center to be proportional to these numbers.

In actuality, some business centers are more active at strange hours than others, but we assume all business hours are proportionally active as a simplifying assumption for the model.

I.5 Proportion of Offsite Repairs

We then needed to find the total proportion of requests that needed offsite repairs. This could be done similarly to above, by finding the total number of offsite repairs in the sample collected data and dividing by the number of data points. After doing this, we found that approximately 0.1805 per repair needed to happen offsite, and a 95% confidence interval for that percentage to be $(0.1716, 0.1894)$.

I.6 Time to Swap Copiers

As for the distributions for the swap times, we are given them as information from the employees as:

- dispatch center $\sim$ Triangular(min=10, max=25, mostlikely=15)
- customer location $\sim$ Triangular(min=20, max=60, mostlikely=30)

I.7 Travel Time

We then needed to model the amount of time it took to get from one location to another (travel time). This was done rather straightforwardly. We had a map of the locations, and the Manhattan distance between each of any two locations. We knew that each van had an average rate of 60kph, and so we could divide to calculate the total amount of travel time. The calculated statistics are as below:

Center	Distance from DC	Travel Time (hours)
BC_1	45	0.75
BC_2	45	0.75
BC_3	35	0.583333333
BC_4	50	0.833333333
BC_5	55	0.916666667
BC_6	25	0.416666667
BC_7	35	0.583333333
BC_8	30	0.5
BC_9	35	0.583333333
BC_10	45	0.75

Table 4: *Map data, tabularized.*

II. Further Explanation of Mathematical Model

II.1 A High Level Explanation

Our Python model simulates the generation of repair requests at a different rate over the course of the day. We have a clock in hours that is set to 0 hours initially. The simulation we carry out is for 24 hours and stops once requests go beyond the 24 hour mark. From our data analysis section we found the average rate per hour requests arrive over the course of the 24 hour day in three hour intervals. Based off of the time, the arrival rate changes. Table 5 shows the arrival rates we used for the Toner It Down! work day.

Interval Start	Interval End	Arrival Rate
12:00 AM	2:59 AM	0.97222
3:00 AM	5:59 AM	3.23333
6:00 AM	8:59 AM	5.23889
9:00 AM	11:59 AM	6.31111
12:00 PM	2:59 PM	6.42222
3:00 PM	5:59 PM	5.61667
6:00 PM	8:59 PM	3.49444
9:00 PM	11:59 PM	1.3222

Table 5: *Rate table of arrivals.*

From the raw data we are able to find the probability at which a request comes from each business center. This is necessary to simulate proportional arrivals from each business center. The following table includes the CDF of arrival requests from the business centers.

Business Center	CDF
1	0.0344
2	0.1215
3	0.2304
4	0.3621
5	0.4814
6	0.5359
7	0.6635
8	0.7204
9	0.8553
10	1.0

Table 6: *CDF of request proportions from each business center.*

We generate requests with the business center probability and arrival rate parameters. The requests are kept in an ordered event list by the time of each request. We define the mechanics, vans and requests. We define the number of mechanics and vans that are available for circulation which will be adjusted in order to decrease time to repair copiers and time to diagnosis. Once the mechanic and van objects are generated we store them in an array. Each mechanic object has a home base parameter that it originates from, a busy status that keeps track of if it is servicing a request or not, a current location parameter that keeps track of where the mechanic is servicing and a num parameter that is the number of the mechanic. Each van object also has the same parameters as the mechanics.

We look at the first request in the event list and then see if a mechanic is available from the mechanic array. If there are no mechanics available in the whole array then we append the request to an ordered mechanic queue that keeps track of the next request that needs to be attended to. If there are mechanics available we decide which mechanic to send. We find the distance between all mechanics available to the request and send the closest one.

Once the mechanic is assigned to the request, its location is updated to the business center it is attending to. The mechanic takes time to travel to the business center. The travel time it takes for the mechanic to arrive is based off of the distance and the speed of the vehicle. Table 7 shows the time it takes to get from the dispatch center to each of the business centers. Table 8 shows the time it takes to get between all of the business centers. These times are stored in the model and are retrieved to calculate how long it takes for requests to be filled.

We are given that the vans travel at 60kph, and we display the below travel times in hours.

Center	Distance from DC	Travel Time (hours)
BC_1	45	0.75
BC_2	45	0.75
BC_3	35	0.583333333
BC_4	50	0.833333333
BC_5	55	0.916666667
BC_6	25	0.416666667
BC_7	35	0.583333333
BC_8	30	0.5
BC_9	35	0.583333333
BC_10	45	0.75

Table 7: *Map data, tabularized.*

Business Centers	Distance	Travel Time
(4,5), (6,8), (8,9)	5	0.08333
(2,3), (6,7), (6,9), (7,10), (9,10)	10	0.16667
(3,4), (7,8), (8,10)	15	0.25
(1,2), (7,9), (3,5), (6,10), (7,9)	20	0.333
(2,4)	25	0.41667
(1,3), (2,5)	30	0.5
(3,6)	40	0.667
(1,4), (3,8)	45	0.75
(1,5), (1,6), (2,6), (3,7), (3,9)	50	0.8333
(1,8), (2,8), (4,6)	55	0.916667
(1,7), (1,9), (4,8), (3,10), (2,7), (2,9), (5,6)	60	1
(4,7), (4,9), (5,8)	65	1.08333
(1,10), (2,10), (5,9), (5,7)	70	1.16667
(4,10)	75	1.25
(5,10)	80	1.333

Table 8: *Distance between business center pairs.*

Once the mechanic arrives at the business center the service time of the initial diagnosis depends upon the raw data we were given. As shown in the Data Analysis section, we found the mean and standard deviation for the diagnosis time of each of the business centers. The diagnosis times are normally distributed with mean 16.13 and standard deviation 2.89 for business center 1, 4, 5, 6, 7, 8, and 10. For the rest of the business centers, the diagnosis time is normally distributed with mean 22.15 and 5.34. These parameters are all in hours. The mechanic time, diagnosis time and time it takes for on site repair are all added to together when the request is taken by the mechanic. This initial response time is appended to an initial response array. The mean of this array is taken once the simulation ends to find the average initial response time.

Once the initial diagnosis is determined, the mechanic must do an on site repair or replace the printer. In the Data Analysis section we found the proportion of requests that needed offsite repairs to be 0.1805 and therefore the percentage of on site repairs to be 1-0.1805.

The onsite repair time was a Beta distribution with parameters (2.6158, 7.4606, -0.4622, 93.475) as determined in our Data Analysis section where we graphed the raw onsite repair time and it represented a Beta distribution.

Once the clock has reached the onsite repair completion time, the mechanic's status is updated to available. Then the mechanic queue is checked and the mechanic attends to the next request. If there are no mechanic requests in the queue the mechanic stays at the business center until the mechanic queue has a request.

For offsite repairs, the mechanic must request a van in order to bring a new copier to the business center and take away the broken copier. We look through the van array to see if a van is available. If there are no vans available in the whole list then we append the request to an ordered van queue that keeps track of the next offsite request that needs to be attended to. If there are vans available we send one. All vans come from the dispatch center so the time it takes to get to the business center does not matter.

The time it takes the van to get to the business center is retrieved from the data that has been stored in the model. The time it takes to replace a broken copier is Triangular(min=20, max=60, most likely=30) according to the Data Analysis section. Once the broken copier is replaced with a new one, the van and the mechanic take the same time to get back to the dispatch center. The time it takes to off load the broken printer is Triangular(min=10,max=25, most likely=15) according to the Data Analysis section. The time it takes for the van to travel to the business center, replace the copier and travel back to the dispatch center is determined when the van is assigned to the request. The time it takes for the mechanic to arrive and diagnose the request is added to the time it takes the van to travel to and replace the copier in order to find the total time it takes to replace a broken copier. This time is appended to an array that keeps track of the time to fix a broken copier. At the end the mean of this array is taken to find the average time to replace a copier.

Once the broken copier is loaded off the van, the van's status is updated to available. Then the van queue is checked and the mechanic attends to the next request. If there are no van requests in the queue the van waits at the dispatch center until a request comes up in the van queue. Figure 16 serves as a flowchart/high level diagram as to how our model works and what our modeling assumptions were.

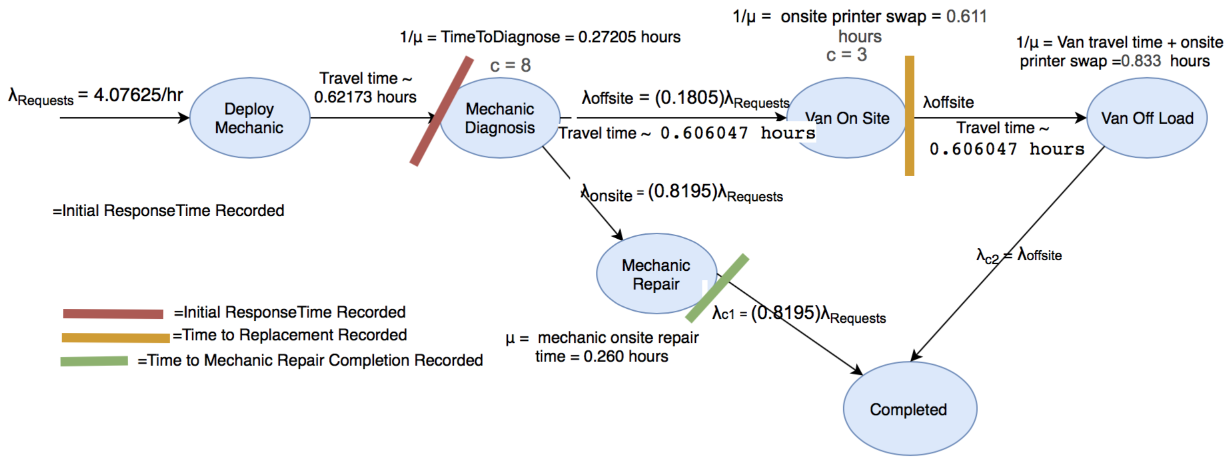

λRequests = 4.07625/hr
Deploy Mechanic
Travel time ~ 0.62173 hours
1/μ = TimeToDiagnose = 0.27205 hours
c = 8
Mechanic Diagnosis
λoffsite = (0.1805)λRequests
Travel time ~ 0.606047 hours
1/μ = onsite printer swap = 0.611 hours
c = 3
Van On Site
λoffsite
Travel time ~ 0.606047 hours
1/μ = Van travel time + onsite printer swap =0.833 hours
Van Off Load
=Initial ResponseTime Recorded
λonsite = (0.8195)λRequests
Mechanic Repair
λc2 = λoffsite
λc1 = (0.8195)λRequests
μ = mechanic onsite repair time = 0.260 hours
Completed
=Initial ResponseTime Recorded
=Time to Replacement Recorded
=Time to Mechanic Repair Completion Recorded

Figure 16: *Flowchart representing the model logic.*

III. Sensitivity Analysis Data

Function	New Parameter	Avg Response Time	CI for Response Time	Avg. Replacement Time	CI for Replacement Time
Diagnosis Time - Mean 22	*N*(19.935, 5.34)	0.64238	(0.59638, 0.688388)	2.603945	(2.49294,2.71494)
Diagnosis Time - Mean 22	*N*(22.15, 4.806)	0.609196	(0.568196, 0.650196)	2.584427	(2.47642, 2.692427)
Diagnosis Time - Mean 16	*N*(14.517, 2.89)	0.55532	(0.52032, 0.59032)	2.475069	(2.37906,2.57106)
Diagnosis Time - Mean 16	*N*(16.13, 2.601)	0.59468	(0.55368, 0.63568)	2.55995	(2.45095, 2.66895)
Repair Time on Site	*Beta*(2.35422, 7.4606, -.4622,93.475)	0.56955	(0.52755, 0.611551)	2.50254	(2.39854,2.60654)
Repair Time on Site	*Beta*(2.6158, 6.71454,-.4622 ,93.475)	0.656676	(0.60367,0.709676)	2.65834	(2.528345, 2.788345)
Repair Time on Site	*Beta*(2.6158, 7.4606,-0.41598,93.475)	0.647626	(0.60062,0.694626)	2.60686	(2.48986, 2.723865)
Repair Time on Site	*Beta*(2.6158, 7.4606,-.4622 ,84.1275)	0.55146	(0.51146, 0.591466)	2.4629	(2.35890, 2.566902)
Arrival Rates	[.837, 3.233, 5.239, 6.311, 6.422, 5.617, 3.494, 1.322]	0.6150751	(0.57507, 0.655075)	2.57561	(2.477614,2.6736145)
Arrival Rates	[.972, 3.006, 5.239, 6.311, 6.422, 5.617, 3.494, 1.322]	0.65505	(0.60805,0.70205)	2.5907585	(2.4747585,2.7068585)
Arrival Rates	[.972, 3.233,4.919, 6.311, 6.422, 5.617, 3.494, 1.322]	0.0611011	(0.568011,0.65401)	2.54005	(2.43405,2.64605)
Arrival Rates	[.972, 3.233, 5.239, 5.943, 6.422, 5.617, 3.494, 1.322]	0.566176	(0.52717,0.60517)	2.45454	(2.35654,2.552550
Arrival Rates	[.972, 3.233, 5.239, 6.311, 6.041, 5.617, 3.494, 1.322]	0.542963	(0.50496,0.58096)	2.48361	(2.375610,2.59161)
Arrival Rates	[.972, 3.233, 5.239, 6.311, 6.422, 5.254, 3.494, 1.322]	0.67771	(0.62871,0.72671)	2.64403	(2.52703,2.76103)
Arrival Rates	[.972, 3.233, 5.239, 6.311, 6.422, 5.617, 3.254, 1.322]	0.622757	(0.57875, 0.666757)	2.519045	(2.422045, 2.616045)
Arrival Rates	[.972, 3.233, 5.239, 6.311, 6.422, 5.617, 3.494, 1.181]	0.613565	(0.570565, 0.65656)	2.56473	(2.450732, 2.678732)

Table 9: *Data collected using/adjusting the lower bounds of the 95% confidence interval for each parameter (in hours).*

Function	New Parameter	Avg. Response Time	CI for Response Time	Avg. Replacement Time	CI for Replacement Time
Diagnosis Time - Mean 22	*N*(24.365 5.34)	0.62535	(0.58235, 0.66835)	2.55026	(2.442267,2.658267)
Diagnosis Time - Mean 22	*N*(22.15, 5.874)	0.611339	(0.565339, 0.657339)	2.566808	(2.454808, 2.678808)
Diagnosis Time - Mean 16	*N*(17.743, 2.89)	0.6621836	(0.61818, 0.706183)	2.62741	(2.519417,2.735417)
Diagnosis Time - Mean 16	*N*(16.13, 3.179)	0.6180858	(0.572085, 0.664085)	2.541736	(2.43573, 2.64773)
Repair Time on Site	*Beta*(2.87738, 7.4606, -.4622,93.475)	0.66065	(0.61565, 0.70565)	2.69166	(2.57666,2.80666)
Repair Time on Site	*Beta*(2.6158, 8.20666,-0.4622 ,93.475)	0.612452	(0.56845,0.656452)	2.573544	(2.466544, 2.680544)
Repair Time on Site	*Beta*(2.6158, 7.4606,-0.50842,93.475)	0.60161	(0.55961,0.643610)	2.48999	(2.390993, 2.58899)
Repair Time on Site	*Beta*(2.6158, 7.4606,-.4622 ,102.8225)	0.707723	(0.65872, 0.756723)	2.74774	(2.62974, 2.865746)
Arrival Rates	[1.107, 3.233, 5.239, 6.311, 6.422, 5.617, 3.494, 1.322]	0.605698	(0.560698, 0.650698)	2.6172	(2.501202,2.733202)
Arrival Rates	[.972, 3.461, 5.239, 6.311, 6.422, 5.617, 3.494, 1.322]	0.595426	(0.555426,0.635426)	2.56404	(2.455046,2.673046)
Arrival Rates	[.972, 3.233,5.558, 6.311, 6.422, 5.617, 3.494, 1.322]	0.633461	(0.586461,0.680461)	2.576064	(2.469064,2.683064)
Arrival Rates	[.972, 3.233, 5.239, 6.679, 6.422, 5.617, 3.494, 1.322]	0.64464	(0.600642,0.688642)	2.60755	(2.503552,2.71155)
Arrival Rates	[.972, 3.233, 5.239, 6.311, 6.804, 5.617, 3.494, 1.322]	0.664575	(0.616575,0.712575)	2.632515	(2.520515,2.744515)
Arrival Rates	[.972, 3.233, 5.239, 6.311, 6.422, 5.979, 3.494, 1.322]	0.689241	(0.639241,0.739241)	2.679923	(2.55492,2.804923)
Arrival Rates	[.972, 3.233, 5.239, 6.311, 6.422, 5.617, 3.736, 1.322]	0.627312	(0.587312, 0.667312)	2.60129	(2.493298, 2.709298)
Arrival Rates	[.972, 3.233, 5.239, 6.311, 6.422, 5.617, 3.494, 1.463]	0.64618	(0.600181, 0.692181)	2.614055	(2.510055, 2.7180550)

Table 10: *Data collected using/adjusting the upper bounds of the 95% confidence interval for each parameter (in hours).*

IV. Simulation Code

```python
import numpy as np
import scipy.stats as st
import pandas

# timeBetweenBCs[i][j] is the amount of time in hours it takes to get from BC i to BC j.
timeBetweenBCs = {
  '0':{'0': 0,'1': 3/4,'2':3/4,'3':7/12,'4':5/6,'5':11/12, '6': 5/12, '7':7/12, '8':1/2, '9':7/12, '10':3/4, 'A':2/3,'B':1/6},
  '1':{'1': 0,'2':1/3,'3':1/2,'4':3/4,'5':5/6, '6': 5/6, '7':1, '8':11/12, '9':1, '10':7/6, 'A':2/3,'B':1/6},
  '2':{'1': 1/3,'2':0,'3':1/6,'4':5/12,'5':1/2, '6': 5/6, '7':1, '8':11/12, '9':1, '10':7/6,'A':1/3, 'B':1/6},
  '3':{'1': 1/2,'2':1/6,'3':0,'4':1/4,'5':1/3, '6': 2/3, '7':5/6, '8':3/4, '9':5/6, '10':1, 'A':1/6, 'B':1/2},
  '4':{'1': 3/4,'2':5/12,'3':1/4,'4':0,'5':1/12, '6': 11/12, '7':13/12, '8':1, '9':13/12, '10':1.25, 'A':1/12,'B':7/12},
  '5':{'1': 5/6,'2':1/2,'3':1/3,'4':1/12,'5':0, '6': 1, '7':7/6, '8':13/12, '9':7/6, '10':4/3,'A':1/6, 'B':2/3},
  '6':{'1': 5/6,'2':5/6,'3':2/3,'4':11/12,'5':1, '6': 0, '7':1/6, '8':1/12, '9':1/6, '10':1/3,'A':5/6,'B':2/3},
  '7':{'1': 1,'2':1,'3':5/6,'4':13/12,'5':7/6, '6':1/6, '7':0, '8':0.25, '9':1/3, '10':1/6, 'A':1,'B':1/12},
  '8':{'1': 11/12,'2':11/12,'3':3/4,'4':1,'5':13/12, '6':1/12, '7':0.25, '8':0, '9':1/12, '10':0.25,'A':11/12,'B':3/4},
  '9':{'1': 1,'2':1,'3':5/6,'4':13/12,'5':7/6, '6':1/6, '7':1/3, '8':1/12, '9':0, '10':1/6, 'A':1,'B':5/6},
  '10':{'1': 7/6,'2':7/6,'3':1,'4':5/4,'5':4/3, '6':1/3, '7':1/6, '8':0.25, '9':1/6, '10':0, 'A':7/6,'B':1},
  'A': {'1':2/3, '2': 1/3, '3':1/6, '4':1/12,'5':1/6, '6':5/6, '7':1, '8':11/12,'9':1, '10':7/6},
  'B': {'1':1/6, '2': 1/6, '3':1/2, '4':7/12,'5':2/3, '6':2/3, '7':1/12, '8':3/4,'9':5/6, '10':1}
}

class Van:
    """
    A class to represent a van.
    """

    def __init__(self,number):
        """
        Initializes a van with ID number number.
        """
        self.homeBase = 0
        self.isBusy = False
        self.currentLocation = 0
        self.num = number

    def changeBusyStatus(self, status):
        """
        Changes the busy status of this van to status.
        If status matches the current busy status of this van, this method throws an exception.
        """
        if status == self.isBusy:
            raise Exception('status already set to this')
        self.isBusy = status

    def changeLocation(self, newLocation):
        """
        Changes the location of this van to newLocation.
        If newLocation matches the current location of this van, this method throws an exception.
        """
        if newLocation == self.currentLocation:
            raise Exception('location is already set to this')
        self.currentLocation = newLocation

class Mechanic:
    """
    A class to represent a mechanic.
    """

    def __init__(self, number, homeBase):
        """
        Initializes a mechanic with ID number number, and home base homeBase.
        """
        self.home = homeBase
        self.isBusy = False
        self.currentLocation = homeBase
        self.num = number

    def changeMechStatus(self,status):
        """
        Changes the busy status of this mechanic to status.
        If status matches the current busy status of this mechanic, this method throws an exception.
        """
        if status == self.isBusy:
            raise Exception('status already set to this')
        self.isBusy = status

    def changeLocation(self,newLocation):
        """
        Changes the location of this mechanic to newLocation.
        """
        self.currentLocation = newLocation

class Request:
    """
    A class to represent an arrival request.
    """

    def __init__(self, TimeAppears):
        """
        Creates a request with time of generation TimeAppears.
        """
        self.BC = getBusinessCenter()
        self.TimeRequested = TimeAppears
        self.InitialResponseTime = None
        self.TimeToCopierReplaced = None

#Helper Functions

def getBusinessCenter():
    """
    Generates a business center with probability proportional to that derived
    from the input distributions.
    """
    Centers = [1,2,3,4,5,6,7,8,9,10]
    Probabilities = [0.0394,0.0821,0.1089,0.1317,0.1193,0.0545,0.1276,0.0569,0.1349,0.1447]
    BC = np.random.choice(Centers, p = Probabilities)
    return BC

def ComputeTravelTime(requestBusinessCenter, originLocation):
    """
    Returns: the amount of time in hours it takes to travel from
    requestBusinessCenter to originLocation.
    """
    time = timeBetweenBCs[str(originLocation)][str(requestBusinessCenter)]
    return time

def generateNextRequest(currentTime):
    """
    Returns: A randomly generated interarrival time for the next arrival based
    on the current time.
    """
    #ArrivalRates
    Rates = [1.107,3.233,5.239,6.311,6.422,5.617,3.494,1.322]
    rate = Rates[int(currentTime // 3)]
    interArrivalTime = np.random.exponential(1 / rate)
    return interArrivalTime

def getDiagnosisTime(BC):
    """
    Returns: A pseudorandom diagnosis time for business center BC.
    """
    params = (22.15, 5.34) if BC in [2, 3, 9] else (16.13, 2.89) # 22.15, 5.34
    time = np.random.normal(*params)
    return time / 60 #convert to hours

def dealWithMechanicRequest(Event):
    """
    Called when a request for service comes in from a business center and
    TID Inc. needs to arrange to send a mechanic. The function checks to see
    whether or not a mechanic is available and deploys the closest mechanic, updates
    the mechanic's status to busy if yes; otherwise adds the request to the queue of
    requests waiting for an available mechanic.
    """
    request = Event[2]
    #Filter to find available mechanics
    AvailableMechanics = [mech for mech in Mechanics if not mech.isBusy]

    if len(AvailableMechanics) == 0: #then no mechanics are available right now
        MechanicQueue.append(Event)
        MechanicQueue.sort(key=lambda x: x[1])

    else: #At least 1 mechanic is free
        bestMechanic = AvailableMechanics[0]
        bestTravelTime = 100000
        for mechanic in AvailableMechanics:
            #check if mechanic is closer
            time = ComputeTravelTime(request.BC,mechanic.currentLocation)
            if time < bestTravelTime:
                bestMechanic = mechanic
                bestTravelTime = time
        bestMechanic.changeMechStatus(True)
        deployMechanic(bestMechanic,Event,bestTravelTime)


def deployMechanic(mechanic,Event,traveltime):
    """
    Called when a mechanic is sent to a business center to start the diagnosis.
    The function updates the mechanic's location according to the business center
    it is servicing and adds an event to the event list that includes the time
    at which the mechanic will be finished travelling to the client, diagnosing the
    issue and (potentially) repairing a printer.
    """
    request = Event[2]
    mechanic.changeLocation(request.BC)
    DiagnosisTime = getDiagnosisTime(request.BC)

    InitialResponseTime = T + traveltime - Event[1]
    InitialResponseTimes.append(InitialResponseTime)
    request.InitialResponseTime = InitialResponseTime

    # Check if need on-site repair, or if need a van.
    # 0 indicates an on site repair and 1 indicates the need for a full printer replacement
    OnSiteOrVan = st.bernoulli.rvs(0.1805)
    if OnSiteOrVan == 0:
        timeToRepairOnSite = st.beta.rvs(2.6158,7.4606,-0.4622,93.475)/60
        EventList.append(['M',T+traveltime+DiagnosisTime+timeToRepairOnSite,mechanic])
        EventList.sort(key=lambda x: x[1])
    else:
        EventList.append(['M',T+traveltime+DiagnosisTime,mechanic])
        EventList.sort(key=lambda x: x[1])
        dealWithVanRequest(Event)


def dealWithVanRequest(Event):
    """
    Called when the mechanic discovers that off-site repair is needed.
    The function checks to see whether or not a van is available and deploys
    the van and changes its status to busy if yes, otherwise adds the request
    to the queue of requests waiting for an available van.
    """
    #Filter to find available vans
    AvailableVans = [van for van in Vans if not van.isBusy]
    if len(AvailableVans) == 0: #then no vans are available right now
        VanQueue.append(Event)
        VanQueue.sort(key=lambda x: x[1])
    else: #At least 1 van is at the headquarters and is equipped with a new copier!
        van = AvailableVans.pop(0)
        van.changeBusyStatus(True)
        deployVan(van,Event)


def deployVan(van,Event):
    """
    Called when a van is sent to a business center to replace a printer. The function
    also adds an event to denote when the van will be available again which happens
    after the time it takes for the van to travel to the business center, replace the
    printer, travel back to the dispatch center and offload the printer.
    """
    timeToReplacement = 0
    TravelTimeOneWay = ComputeTravelTime(Event[2].BC, 0)
    ReplacementTimeAtBusinessCenter = np.random.triangular(20,30,60)/60
    OffloadPrinterTime = np.random.triangular(10,15,25)/60
    FinishTime = T + TravelTimeOneWay + ReplacementTimeAtBusinessCenter
    FinishTime += TravelTimeOneWay + OffloadPrinterTime
    EventList.append(['V', FinishTime,van])
    EventList.sort(key=lambda x: x[1])
    timeToReplacement = T + TravelTimeOneWay + ReplacementTimeAtBusinessCenter - Event[1]
    DeliveryTimesForReplacement.append(timeToReplacement + Event[2].InitialResponseTime)

#MAIN SIMULATION CODE
numberMechanics = 8
numberVans = 3
numreps = 250

#Total Statistics To Keep Track Of from Multiple Simulation Runs
AverageInitialResponseTimes = np.zeros(numreps)
AverageDeliveryTimesForReplacedCustomers = np.zeros(numreps)

for rep in range(numreps):
    #Stats for a single simulation
    InitialResponseTimes = []
    DeliveryTimesForReplacement = []

    #Create vans and mechanics, adjust vans and mechanics

    Vans = [Van(i) for i in range(numberVans)]
    Mechanics = [Mechanic(i, 0) for i in range(numberMechanics)]

    i = 0
    T = 0
    MechanicQueue = []
    VanQueue = []
    EventList = []
```

```python

    #Initialize with the first arrival
    interArrival = np.random.exponential(1/0.9722)
    firstRequest = Request(interArrival)
    EventList.append(['R', interArrival, firstRequest])

    #Add stopping criteria
    stop = ['S', 24]
    EventList.append(stop)

    #Sort List According to Event Times
    EventList.sort(key=lambda x: x[1])

    while T <= 24:
        Next = EventList.pop(0)
        T = Next[1]

        #CASE 1: If this is a request that comes in
        if Next[0] == 'R':
            #Generate next request
            t = generateNextRequest(T)
            EventList.append(['R',T+t,Request(T+t)])
            EventList.sort(key=lambda x: x[1])
            dealWithMechanicRequest(Next)

        #CASE 2: a mechanic finishes serving a client and therefore becomes available
        if Next[0] == 'M':
            i = i+1
            mechanic = Next[2]
            #Check Mechanic's queue:
            if len(MechanicQueue) > 0: #Mechanic can immediately go to a new client
                requestFromQueue = MechanicQueue.pop(0)
                timeToTravelToNextClient = ComputeTravelTime(requestFromQueue[2].BC, mechanic.currentLocation)
                deployMechanic(mechanic,requestFromQueue,timeToTravelToNextClient)

            else: #Mechanic becomes free and will wait at it's current location
                mechanic.changeMechStatus(False)

        #CASE 3: a van finishes with a client, returns to the dispatch center and offloads printer and therefore is now available
        if Next[0] == 'V':
            van = Next[2]
            #Check Van's queue
            if len(VanQueue) > 0: #Van can immediately go to a new client
                requestFromQueue = VanQueue.pop(0)
                deployVan(van, requestFromQueue)
            else: #Van becomes free and will wait at it's current location
                van.changeBusyStatus(False)

        if Next[0] == 'S':
            break

        #Update Statistics
        avg1 = np.mean(InitialResponseTimes)
        AverageInitialResponseTimes[rep] = avg1

        avg2 = np.mean(DeliveryTimesForReplacement)
        AverageDeliveryTimesForReplacedCustomers[rep] = avg2

#PRINT OUT SOME STATS

#Compute Averages of the Averages

print("Average response time = ",  np.mean(AverageInitialResponseTimes))
avg = np.mean(AverageInitialResponseTimes)
```

```
z = st.norm.ppf(0.975)
offset = z * np.std(AverageInitialResponseTimes, ddof = 1) / np.sqrt(numreps)
print("CI for Average Response Time = ", avg - offset, avg + offset)
print("Average time to replacement = ", np.mean(AverageDeliveryTimesForReplacedCustomers))
response_data_to_plot.append(AverageInitialResponseTimes)
avg = np.mean(AverageDeliveryTimesForReplacedCustomers)
z = st.norm.ppf(0.975)
offset = z * np.std(AverageDeliveryTimesForReplacedCustomers, ddof = 1) / np.sqrt(numreps)
print("CI for Average Replacement = ", avg - offset, avg + offset)
replacement_data_to_plot.append(AverageDeliveryTimesForReplacedCustomers)
```