Charlottesville Airport TSA Queue Simulation

Fan Feng, Jordan Lopez, Justis Midura, and Maxwell Weiss
SYS 3062
Professor Steve Patek and Professor Nicola Bezzo
May 9th, 2017

Abstract

Data was collected on the arrivals and departures of passengers to and from the security checkpoint of the Transportation Security Administration (TSA) at the Charlottesville-Albemarle Airport in order to develop a comprehensive simulation model. This model of the standard security checkpoint will provide for a better understanding of its current structure as well as a thorough analysis of methods to improve the throughput of passengers to a significant extent. Additionally, the simulation model will provide clarity in identifying the particular sub-systems within the security checkpoint as a whole that tend to cause the greatest amounts of congestion on a typical day at the airport. The results from this simulation will facilitate the potential alleviation of congestion within the standard security checkpoint, as well as to provide a recommendation to passengers on the most efficient time to arrive prior to their flight.

Introduction

Throughout the past few years, air transportation has become significantly more common among all age groups around the world. As the world has become more connected, air transportation has become the most effective and efficient way to transport people from one side of the world to the other. It is therefore imperative that the procedure for air transportation be as efficient as possible in regards to time management, as well as provides a strong sense of security for all types of passengers. However, it is fairly well known that within most airports, the security checkpoints have become the most significant time constraint which severely limits the potential throughput of passengers from outside of the airport to their gates. This “bottleneck” dilemma is looked at much more closely within our simulation in order to identify the particular reasons for this occurring. Using Matlab, we employ methods of discrete-event simulation to develop a comprehensive model. The model is treated as a series of different queues that passengers walk through in order to get through the security checkpoint. Throughout our analysis of the security checkpoint at the Charlottesville-Albemarle Airport, we break down the system (being the security checkpoint) into three different sub-systems. These sub-systems are the Pre-Check/General Travel Document Checker (TDC), the conveyor belt for trays, and the walk-through metal detector. These various sub-systems will be discussed in greater depth in the “Existing Infrastructure” section of the paper. It is also important to note that throughout our analysis, various assumptions of the security checkpoint system were made. One obvious assumption was that different levels of congestion within the security checkpoint occur depending on the time until the next flight. Additionally, for the most part, the usage of the Pre-Check or General Travel Document Checker did not have an impact on the throughput of passengers; there was never an incident in which using one of these TDC’s led to an increase in total waiting time for a particular passenger. Lastly, a combination of both observations in different literature (which will be discussed subsequently,) as well as empirical observations at the Charlottesville-Albemarle Airport led us to believe that the walk-through metal detector was the greatest constraint to a large throughput of passengers. Therefore, we decided to pay very close attention to this sub-system within the security checkpoint.

Review of Literature

Following the creation of the TSA in 2001, the average throughput of passengers in the standard security checkpoint decreased by a large margin. This was obviously due to the security procedures TSA implemented upon establishment. Due to this significant impediment in the processing rate of passengers, reducing congestion and alleviating bottlenecks became a very widespread topic of interest and various studies have been conducted to explore it empirically. One particular study pertaining to this simulation model was a study conducted by Tomber and Barros to conclude general methods of increasing passenger throughput of security checkpoints (specifically at Seattle-Tacoma International Airport). Some of the factors analyzed were the number of items allowed per passenger, the physical size of divesting tables, and the use of secondary metal detectors screeners. They found the most impactful factor on throughput to be the number of items allowed per passenger. In 2012, the TSA began to incorporate "divesture officers" in their security checkpoints whose sole purpose was to reduce false alarms of the metal detectors by providing passengers with oral instructions in order to ensure that they divest correctly prior to the screening. Surprisingly, it was found that security checkpoints with these divesture officers have a 9% higher passenger throughput than security checkpoints without them. Another particularly relevant finding was the role of metal detectors in restricting potential throughput within security checkpoints. They have been identified publicly as the main reason for constrained passenger throughput for two reasons: either passengers fail to divest correctly prior to being screened, or, they fail to collect their belongings in a timely manner following their screening. It was also found that the latter reason only pertains to airports with a higher percentage of leisure passengers, therefore, it does not pertain as much with our simulation; we have found that the Charlottesville-Albemarle Airport is a relatively smaller one which passengers use primarily for business travel, rather than for leisure. The solution to the former reason is the same as in the preceding study: incorporate divesture officers who will provide passengers with clear oral instructions on how to divest appropriately. One last relevant study was one conducted in 2006 by the Dallas Fort Worth International Airport. Their planning department studied how extending the exit rollers at the end of the security checkpoint would impact passenger throughput. It was eventually concluded that having the “composure area” (at the end of the exit rollers) farther away from the metal detectors would accomplish two different things: give more space to passengers and thus provide greater comfort, as well as significantly lower total duration of delays and thus diminish number of incidents in which passengers stand near to the exit rollers. From all of these studies and their empirical findings, it can be concluded that various ways to improve average passenger throughput within the security checkpoint at the Charlottesville-Albemarle Airport includes limiting the number of items allowed per passenger, incorporating divesture officers who will provide oral instructions on how to divest appropriately, extending the exit rollers at the end of the security checkpoint, as well as utilizing more metal detectors for screening at once to mitigate the delay caused by passenger error in divesting.

Purpose of Study

At its core, the study seeks to improve the Transportation Security Administration (TSA) queue at Charlottesville-Albemarle Airport. It will do so by identifying states of critical instability, potential bottlenecks in the system, and ways of alleviating traffic when the system is approaching critical instability. For example, it might be found that when a certain amount of people arrive at the first queue while there are already 3 people in the second queue, that the system is at risk of becoming unstable. In this case, we would identify both the number of arrivals in a given time at queue 1 that would lead to instability, and a way to mitigate that risk by, say, potentially opening another baggage belt to alleviate traffic. However, it is also entirely possible that the system is found to function very well. In that case, a recommendation might be to monetize the model Charlottesville has developed for getting people through the TSA checkpoint at airports of a similar size. Ultimately, the objective of the study is to improve the experience for travelers going through CHO (and other airports if that's a possibility.) The study will try to identify a time a traveler must arrive at CHO before a flight to ensure they will make their flight, it will try to find strategies to minimize this time, and it will recommend improvements to the current TSA queue at CHO.

The study will also provide summary statistics of how long a traveler can expect to spend in the system, and potentially strategies that travelers can take to make it faster for themselves and everyone in the system. After all, the system might reach instability because a single traveler just happens to be incredibly slow at doing things in a TSA queue. The more prepared travelers are, the quicker they will move through the baggage check queue and the smaller the delay will be for everyone else.

While the study seeks to identify ways that administrators can speed up their queue and thus improve the experience for travelers flying out of CHO, we understand that TSA is a heavily regulated agency and CHO has limited power in influencing the way TSA handles operations. Furthermore, we realize that a recommendation such as “buy another full body scanner” is unrealistic because these machines cost more than $150,00 per unit (CITE http://www.politico.com/story/2015/08/airport-security-price-for-tsa-failed-body-scanners-160-million-121385), and we have no way of determining the cost/benefit of doing so. However, we think it is feasible for a recommendation to include “TSA should open the second baggage queue if there are X many people in the system” because the CHO TSA checkpoint already has 2 of these, and monitoring how many people in the system is fairly easy to do. If TSA is pre-emptive in opening a second queue, it would be very possible to avoid critical instability (theoretically). A mitigation here, however, is that TSA employees are also a limited resource, so we cannot be positive a recommendation is feasible, necessarily, as it requires many employees to run just one baggage belt queue as it is.

In any case, the purpose of the study is to improve traveler experience by identifying ways in which the process of going through the TSA queue can be sped up.

Existing TSA Infrastructure at CHO

Charlottesville-Albemarle Airport’s TSA checkpoint can be divided into three queues when it is operating normally. For the purposes of the study, we will only focus on normal operations. The first queue is a line that travelers enter when they want to pass through the TSA checkpoint. There are two lines that a traveler can enter before the ID check - pre-check and general boarding - but we found that an insignificant amount of travelers went into pre-check for the system to be affected by it, so as a simplification, we did not include it in the study. The ID check line has a single server -

the person who checks that forms of identification match with the information on boarding passes. If someone is having their identification checked, then new arrivals will form a queue behind this person. There is not too much variability in serving times here because the process is repeated often and all travelers must do is present their boarding pass and IDs.

After a person passes the identification check, they enter the “baggage belt” queue. Here, people remove their shoes, belts, and anything metal, and put their computers individually in bins, as well as their bags and other possessions. This queue has a high degree of variability, because people don’t travel with the same objects (that have different rules and regulations,) and because the speed at which people go through this point depends on experience (how often someone has traveled,) as well as their general speed, and how much they have prepared beforehand. For example, if someone has brought liquids but didn’t put them in a bad ahead of time, they will spend a significant amount of time at this point when they realize the liquids need to be placed in a see-through bag. However, a skilled traveler would have done this beforehand and will go through relatively quickly. For the purposes of this study, we have limited to the amount of people who can use the baggage belt at once to be 3, so the system has 3 independent servers that can serve at the same time. Thus, when someone leaves the ID checkpoint, they will see one of the three “servers” that makes up the baggage check with bins. However, if there are three people there already, then the person cannot leave the ID check until someone has left the baggage point. We know that it’s theoretically possible for more people to use the baggage check belt than 3 at a time, but in our experience at the airport this never happened, so we have capped it at 3 as a simplification.

From the baggage check queue, the travelers form a line into a single server queue that is the metal detector (the big circular machine you have to stand in.) Only one person can be served at a time, but serving times are relatively similar because a TSA agent yells out instructions and then presses a button, and then repeats this. By the time you're in line, you've seen several people do this and you know that all you have to do is stand still with your hands up and wait 3 seconds. For the purposes of the study, once a traveler passes through the metal detector, we consider them out of the system. This is because the time they spend removing their belongings from the bins and putting their clothes back on are truly independent of the rest of the system. For example, if a man spends time tying his shoes, this really doesn't affect how quickly someone makes it through the ID queue. The last point at which a traveler can affect the times other travelers experience is when the are in the metal detector. So once they leave the metal detector, they are "out" of the system. For a graphical representation of the system, view *Figure 1*, below.

As was mentioned previously, CHO has two baggage belts and two metal detectors that can run in parallel, but in our experience, only one of these belts was ever opened, so we did not include the other in the study, but leave the prospect of opening a second one as a "pressure valve" when there is heavy traffic in the system. Additoinally, for photos of the real-life system, please see below.

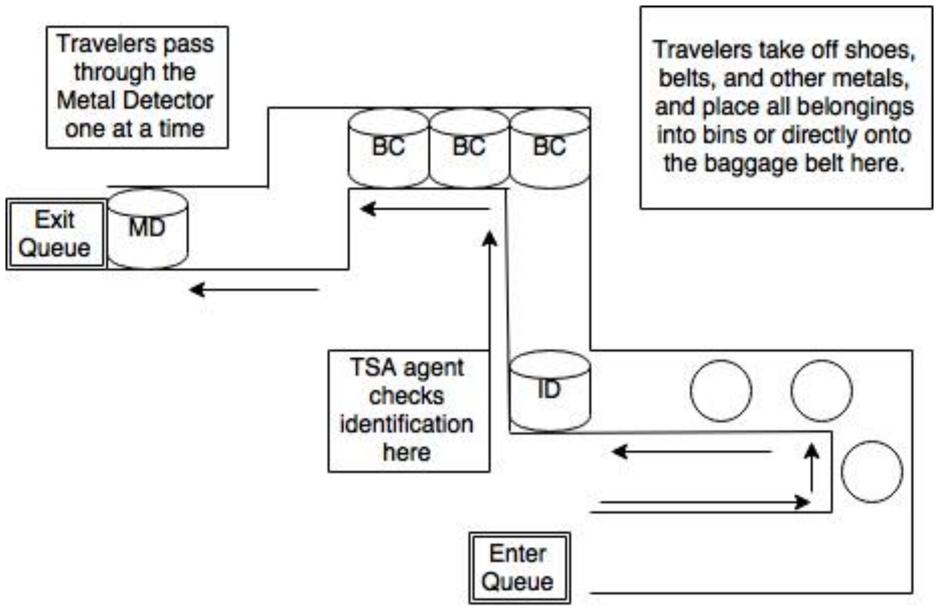

Travelers pass through the Metal Detector one at a time
Travelers take off shoes, belts, and other metals, and place all belongings into bins or directly onto the baggage belt here.
BC
BC
BC
Exit Queue
MD
TSA agent checks identification here
ID
Enter Queue

FIGURE 1 - DIAGRAM OF SYSTEM

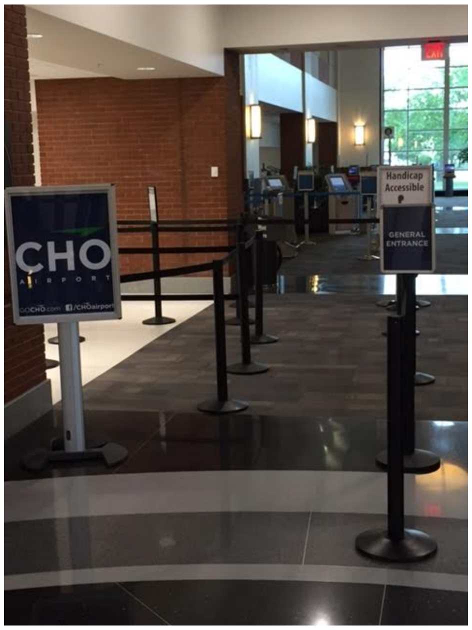


PHOTO 1 - ENTRANCE TO TSA CHECKPOINT

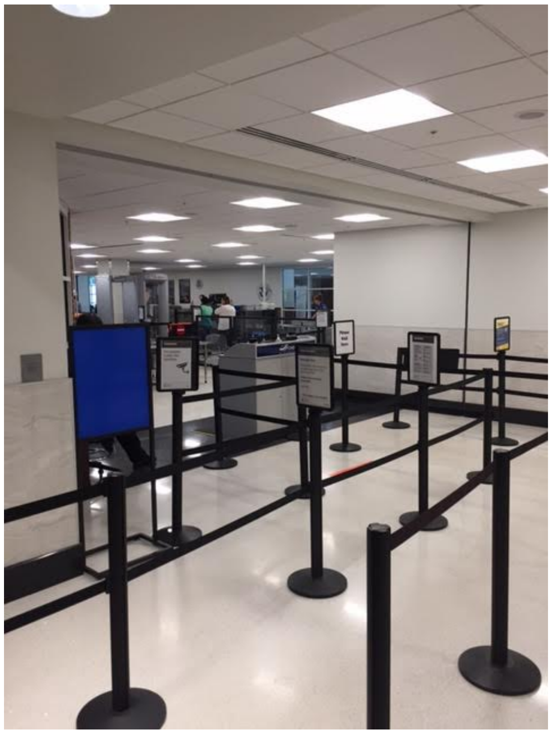


PHOTO 2 - WAITING QUEUE FOR ID CHECK

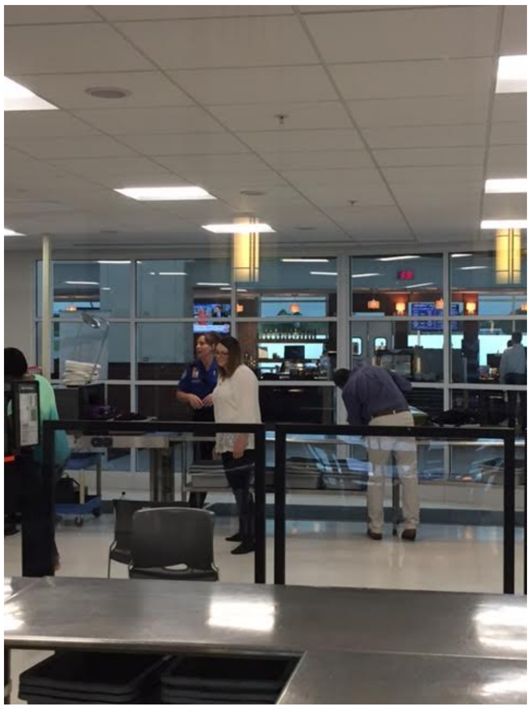


PHOTO 3 - BAGGAGE BELT

PHOTO 3 - BAGGAGE BELT

Methods and Data Collection

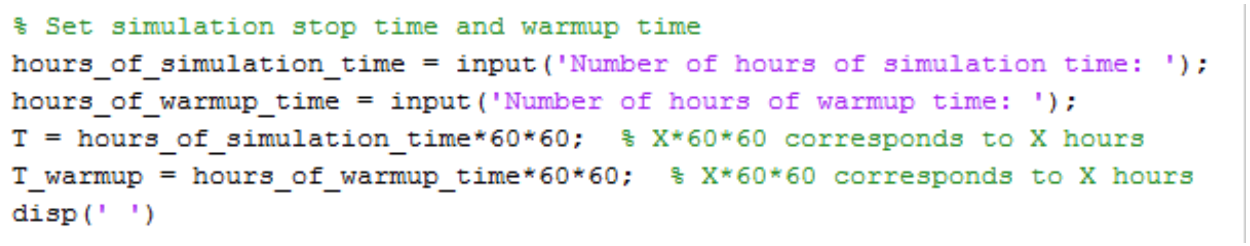
Data was collected on three independent days at the airport. The first day data was collected, it was done manually in excel. Two people had timers (that started at the same time,) and one person had excel open on their computer. One of the timers would tell the times that arrivals occurred at the entire system, and the other would say when departures occurred at the entire system. However, this data was scrapped for several reasons. First, this took place on a Sunday at around 10 AM, which was a non-representative time for traveling at CHO because most people will not be leaving CHO on a Sunday; there was only one flight, so arrivals were extremely infrequent. Furthermore, at this point, the team had not realized the system was actually a system of 3 queues, so our data could not give insight into the true nature of the system.

The second time data was collected, it was collected in the same way but at 6:30-8:30 AM on a Friday. This was a very representative time for traveling, but again, we had not yet realized the system needed to be represented with 3 queues instead of just one. However, we used data taken here to determine the distribution type of inter-arrival times, which was exponential with a very good fit. Similarly, we found that time spent in stem was also exponential. However, only after we met as a group after this data collection session did we realize we hadn't understood the true nature of the system. Though discussion, we realized an accurate representation of the system needed to be in terms of three queues in series. But the way we were collecting data would be too inefficient if we were going to identify the serving times of each of those queues, which would require keeping track of arrivals and departures at each queue. Consequently, we wrote a Matlab script that we could all run at once that allowed us on our respective computers that allowed us to individually track queues.

The third time we went back was also a Friday morning at 6:30 AM, and this time we collected data individually to calculate serving times for each of the queues. Thus, between all data collection days, we were able to construct distributions for each of the 3 queues, as well as an approximate time in system distribution and a distribution for inter-arrivals.

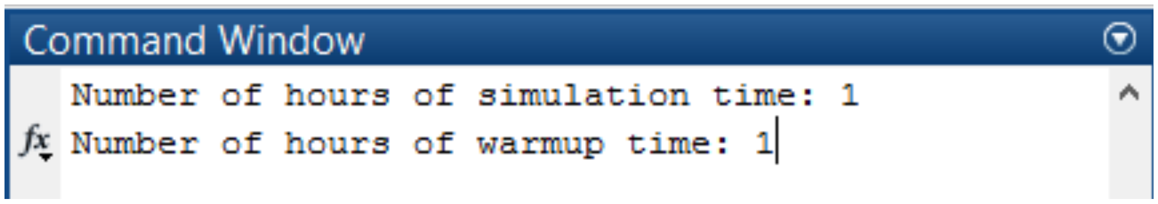
Discussion of Simulation

The MATLAB code to run the simulation is 583 lines long. The code utilizes the exponential random variable function that was provided in example codes from the discrete events simulation class. The code is structured so simulation time and warmup time inputs are collected at the beginning, then all variables are initialized, then there is a massive while loop that goes through 10 possible cases of the simulation until the simulation time is over, and then three output statements and four graphs are created to analyze results.



```
% Set simulation stop time and warmup time
hours_of_simulation_time = input('Number of hours of simulation time: ');
hours_of_warmup_time = input('Number of hours of warmup time: ');
T = hours_of_simulation_time*60*60;  % X*60*60 corresponds to X hours
T_warmup = hours_of_warmup_time*60*60;  % X*60*60 corresponds to X hours
disp(' ')
```

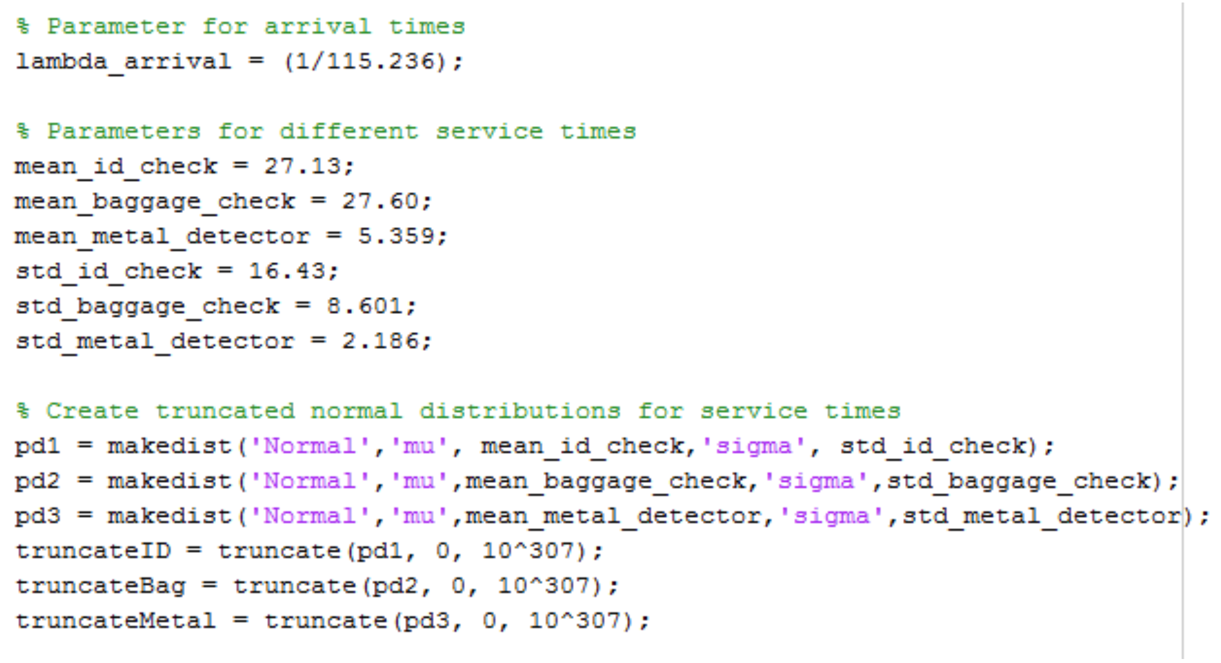
FIGURE 2



```
Number of hours of simulation time: 1
Number of hours of warmup time: 1
```

FIGURE 3

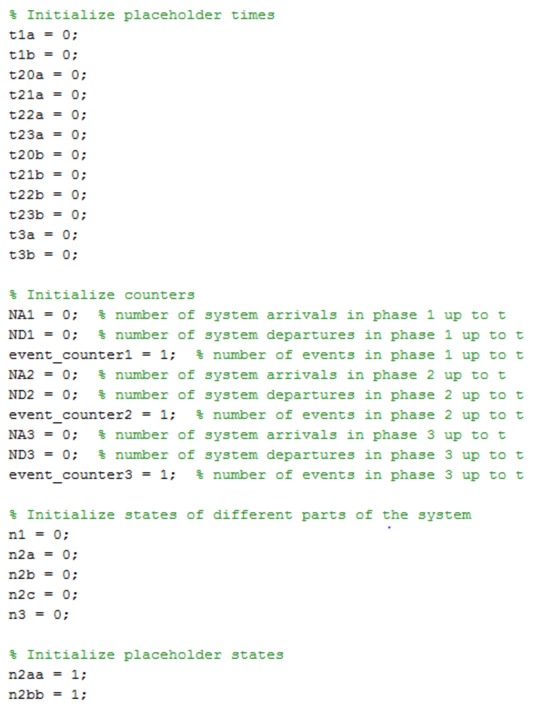
Simulation time and warmup time inputs are collected before the simulation begins in order to know how long to run the simulation before recording data and how long to run the simulation while recording data. The times are collected in hours and then are converted into seconds because the rest of the simulation is coded using seconds, which is simpler.



```
% Parameter for arrival times
lambda_arrival = (1/115.236);

% Parameters for different service times
mean_id_check = 27.13;
mean_baggage_check = 27.60;
mean_metal_detector = 5.359;
std_id_check = 16.43;
std_baggage_check = 8.601;
std_metal_detector = 2.186;

% Create truncated normal distributions for service times
pd1 = makedist('Normal','mu', mean_id_check,'sigma', std_id_check);
pd2 = makedist('Normal','mu',mean_baggage_check,'sigma',std_baggage_check);
pd3 = makedist('Normal','mu',mean_metal_detector,'sigma',std_metal_detector);
truncateID = truncate(pd1, 0, 10^307);
truncateBag = truncate(pd2, 0, 10^307);
truncateMetal = truncate(pd3, 0, 10^307);
```

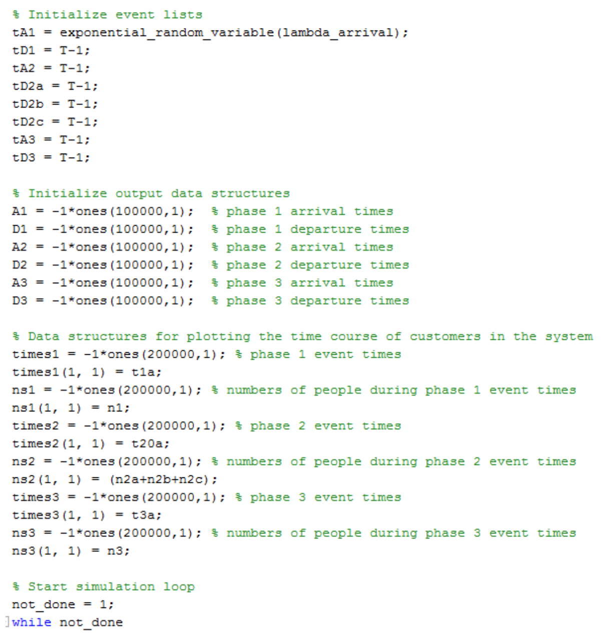
FIGURE 4

The first values to be initialized are the parameters for arrival times and service times for all three phases of the security process. These values were discovered through data collection and can be altered to test performance of the system under different conditions. The arrival process was fit to an exponential distribution while all three phases of the security process were fit to truncated normal distributions. The truncated normal distributions were created by creating the normal distributions with the mean and standard deviation of their respective phase and then truncating the distributions by zero and positive infinity using the truncate function provided by MATLAB.



```
% Initialize placeholder times
t1a = 0;
t1b = 0;
t20a = 0;
t21a = 0;
t22a = 0;
t23a = 0;
t20b = 0;
t21b = 0;
t22b = 0;
t23b = 0;
t3a = 0;
t3b = 0;

% Initialize counters
NA1 = 0;  % number of system arrivals in phase 1 up to t
ND1 = 0;  % number of system departures in phase 1 up to t
event_counter1 = 1;  % number of events in phase 1 up to t
NA2 = 0;  % number of system arrivals in phase 2 up to t
ND2 = 0;  % number of system departures in phase 2 up to t
event_counter2 = 1;  % number of events in phase 2 up to t
NA3 = 0;  % number of system arrivals in phase 3 up to t
ND3 = 0;  % number of system departures in phase 3 up to t
event_counter3 = 1;  % number of events in phase 3 up to t

% Initialize states of different parts of the system
n1 = 0;
n2a = 0;
n2b = 0;
n2c = 0;
n3 = 0;

% Initialize placeholder states
n2aa = 1;
n2bb = 1;
```

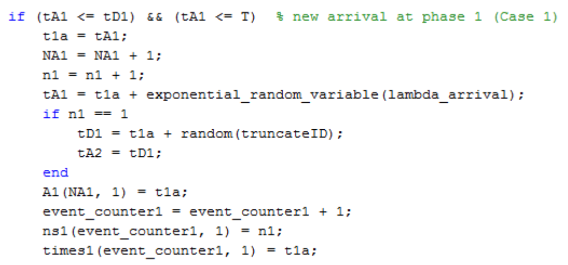
FIGURE 5



```
% Initialize event lists
tA1 = exponential_random_variable(lambda_arrival);
tD1 = T-1;
tA2 = T-1;
tD2a = T-1;
tD2b = T-1;
tD2c = T-1;
tA3 = T-1;
tD3 = T-1;

% Initialize output data structures
A1 = -1*ones(100000,1);  % phase 1 arrival times
D1 = -1*ones(100000,1);  % phase 1 departure times
A2 = -1*ones(100000,1);  % phase 2 arrival times
D2 = -1*ones(100000,1);  % phase 2 departure times
A3 = -1*ones(100000,1);  % phase 3 arrival times
D3 = -1*ones(100000,1);  % phase 3 departure times

% Data structures for plotting the time course of customers in the system
times1 = -1*ones(200000,1); % phase 1 event times
times1(1, 1) = t1a;
ns1 = -1*ones(200000,1); % numbers of people during phase 1 event times
ns1(1, 1) = n1;
times2 = -1*ones(200000,1); % phase 2 event times
times2(1, 1) = t20a;
ns2 = -1*ones(200000,1); % numbers of people during phase 2 event times
ns2(1, 1) = (n2a+n2b+n2c);
times3 = -1*ones(200000,1); % phase 3 event times
times3(1, 1) = t3a;
ns3 = -1*ones(200000,1); % numbers of people during phase 3 event times
ns3(1, 1) = n3;

% Start simulation loop
not_done = 1;
while not_done
```

FIGURE 6

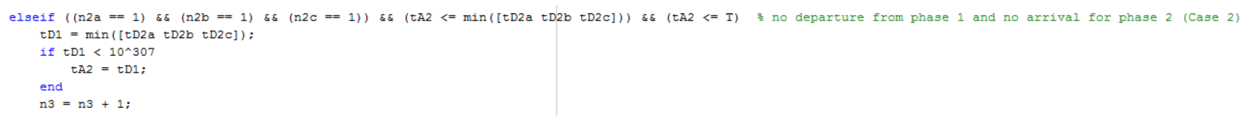
Place holder times are all initialized to zero and are used throughout the loop to hold arrival times and departure times to create new arrival times and departure times and store old values in arrays for analyzing results after the simulation. Counters for total number of arrivals and total number of departures for each phase are initialized to zero as well as event counters for each phase which are used for analyzing results as well. The number of people at each state are initialized to zero because no one starts out in the system. Placeholder states are initialized to one and are used to coordinate which case to execute during a specific instance while in the simulation loop. Temporary arrival time for the first phase is initialized to the first arrival time pulled from the exponential distribution and the rest of the temporary arrival times and departure times are initialized to the simulation time minus one because no one is in the system yet. Output data structures for arrival times and departure times at each phase are initialized to arrays full of negative ones and are used for analyzing results after the simulation. Time and people in the system data structures for each phase are initialized to arrays full of negative ones and are used for analyzing results as well. A variable that flags when to continue running the simulation and when to end the simulation is initialized to one because the simulation starts out running.



```
if (tA1 <= tD1) && (tA1 <= T)  % new arrival at phase 1 (Case 1)
    t1a = tA1;
    NA1 = NA1 + 1;
    n1 = n1 + 1;
    tA1 = t1a + exponential_random_variable(lambda_arrival);
    if n1 == 1
        tD1 = t1a + random(truncateID);
        tA2 = tD1;
    end
    A1(NA1, 1) = t1a;
    event_counter1 = event_counter1 + 1;
    ns1(event_counter1, 1) = n1;
    times1(event_counter1, 1) = t1a;
```

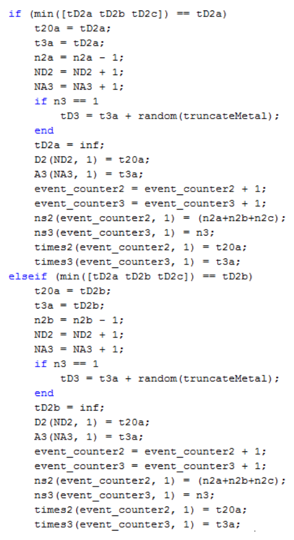
FIGURE 7

The first case is when there is an arrival at phase one and the time is less than the simulation time. One person enters phase one and a new arrival time is calculated for the next person to arrive at phase one. If the person who just entered phase one is the only person in phase one, then a departure time is calculated for that person.



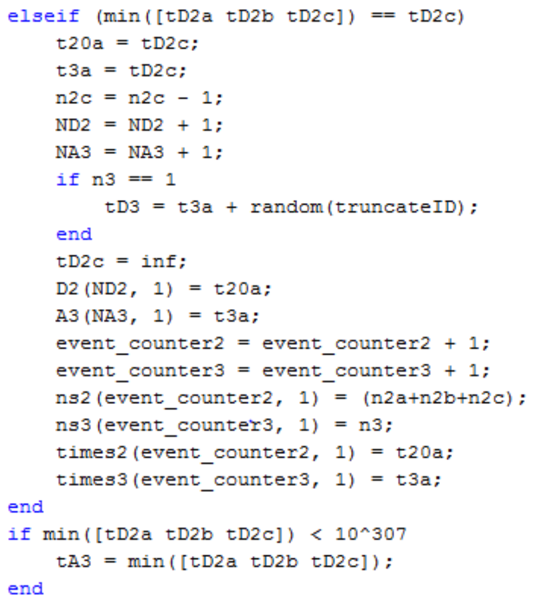
```
elseif ((n2a == 1) && (n2b == 1) && (n2c == 1)) && (tA2 <= min([tD2a tD2b tD2c])) && (tA2 <= T)  % no departure from phase 1 and no arrival for phase 2 (Case 2)
    tD1 = min([tD2a tD2b tD2c]);
    if tD1 < 10^307
        tA2 = tD1;
    end
    n3 = n3 + 1;
```

FIGURE 8



```
if (min([tD2a tD2b tD2c]) == tD2a)
    t20a = tD2a;
    t3a = tD2a;
    n2a = n2a - 1;
    ND2 = ND2 + 1;
    NA3 = NA3 + 1;
    if n3 == 1
        tD3 = t3a + random(truncateMetal);
    end
    tD2a = inf;
    D2(ND2, 1) = t20a;
    A3(NA3, 1) = t3a;
    event_counter2 = event_counter2 + 1;
    event_counter3 = event_counter3 + 1;
    ns2(event_counter2, 1) = (n2a+n2b+n2c);
    ns3(event_counter3, 1) = n3;
    times2(event_counter2, 1) = t20a;
    times3(event_counter3, 1) = t3a;
elseif (min([tD2a tD2b tD2c]) == tD2b)
    t20a = tD2b;
    t3a = tD2b;
    n2b = n2b - 1;
    ND2 = ND2 + 1;
    NA3 = NA3 + 1;
    if n3 == 1
        tD3 = t3a + random(truncateMetal);
    end
    tD2b = inf;
    D2(ND2, 1) = t20a;
    A3(NA3, 1) = t3a;
    event_counter2 = event_counter2 + 1;
    event_counter3 = event_counter3 + 1;
    ns2(event_counter2, 1) = (n2a+n2b+n2c);
    ns3(event_counter3, 1) = n3;
    times2(event_counter2, 1) = t20a;
    times3(event_counter3, 1) = t3a;
```

FIGURE 9

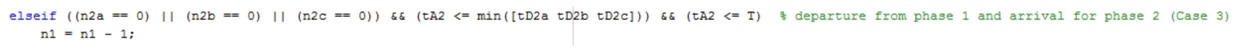


```
elseif (min([tD2a tD2b tD2c]) == tD2c)
    t20a = tD2c;
    t3a = tD2c;
    n2c = n2c - 1;
    ND2 = ND2 + 1;
    NA3 = NA3 + 1;
    if n3 == 1
        tD3 = t3a + random(truncateID);
    end
    tD2c = inf;
    D2(ND2, 1) = t20a;
    A3(NA3, 1) = t3a;
    event_counter2 = event_counter2 + 1;
    event_counter3 = event_counter3 + 1;
    ns2(event_counter2, 1) = (n2a+n2b+n2c);
    ns3(event_counter3, 1) = n3;
    times2(event_counter2, 1) = t20a;
    times3(event_counter3, 1) = t3a;
end
if min([tD2a tD2b tD2c]) < 10^307
    tA3 = min([tD2a tD2b tD2c]);
end
```

FIGURE 10

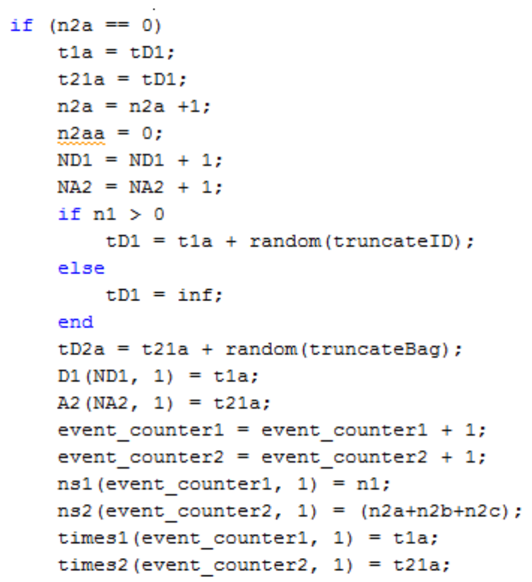
The second case is when a person in phase one is ready to depart and the time is less than the simulation time, but all the lines in phase two are at capacity so they must wait. The departure time for the person waiting in phase one is set to the minimum departure time of each person at phase two because when the first person leaves at phase two then the person at phase one will leave to phase two. Since someone has just left phase two, we add a person to phase three. The first instance in the second case is when the minimum departure time of each person at phase two is the person in the first line. When this is true, the person in the first line leaves and the next departure time is set to infinity because now no one is in the first line. If the person who left the first line is the first person in line for phase three, then the departure time for that person is calculated for phase three. The second instance in the second case is when the minimum departure time of each person at phase two is the person in the second line.

When this is true, the person in the second line leaves and the next departure time is set to infinity because now no one is in the second line. If the person who left the second line is the first person in line for phase three, then the departure time for that person is calculated for phase three. The third instance in the second case is when the minimum departure time of each person at phase two is the person in the third line. When this is true, the person in the third line leaves and the next departure time is set to infinity because now no one is in the third line. If the person who left the third line is the first person in line for phase three, then the departure time for that person is calculated for phase three. The arrival time for phase three is then set to the minimum departure time of each person at phase two.



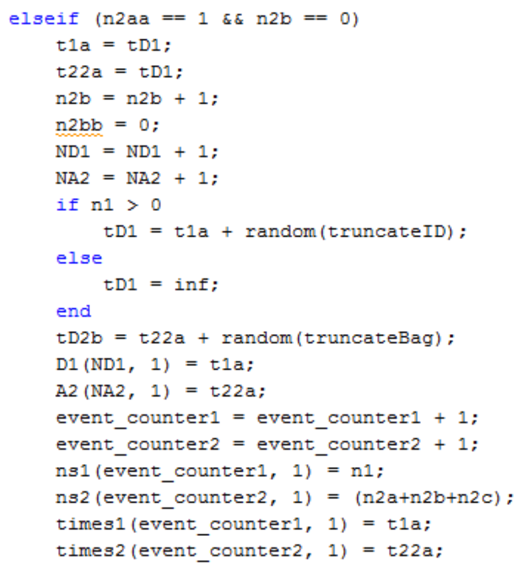
```
elseif ((n2a == 0) || (n2b == 0) || (n2c == 0)) && (tA2 <= min([tD2a tD2b tD2c])) && (tA2 <= T)  % departure from phase 1 and arrival for phase 2 (Case 3)
    n1 = n1 - 1;
```

FIGURE 11



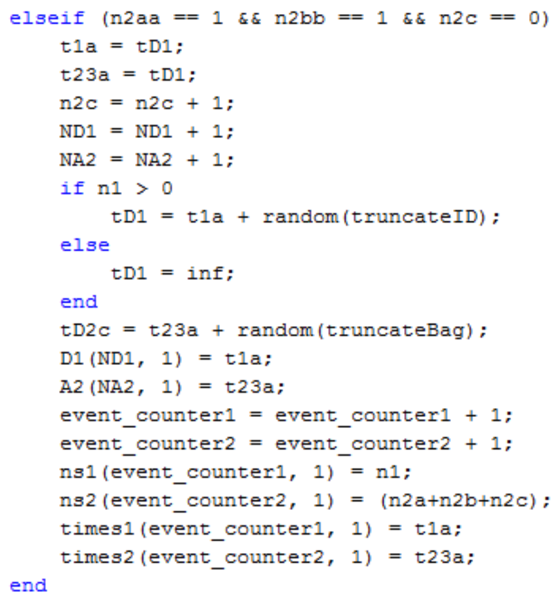
```
if (n2a == 0)
    t1a = tD1;
    t21a = tD1;
    n2a = n2a +1;
    n2aa = 0;
    ND1 = ND1 + 1;
    NA2 = NA2 + 1;
    if n1 > 0
        tD1 = t1a + random(truncateID);
    else
        tD1 = inf;
    end
    tD2a = t21a + random(truncateBag);
    D1(ND1, 1) = t1a;
    A2(NA2, 1) = t21a;
    event_counter1 = event_counter1 + 1;
    event_counter2 = event_counter2 + 1;
    ns1(event_counter1, 1) = n1;
    ns2(event_counter2, 1) = (n2a+n2b+n2c);
    times1(event_counter1, 1) = t1a;
    times2(event_counter2, 1) = t21a;
```

FIGURE 12



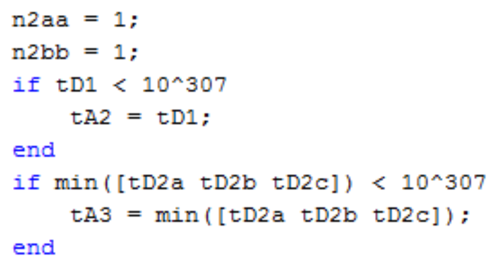
```
elseif (n2aa == 1 && n2b == 0)
    t1a = tD1;
    t22a = tD1;
    n2b = n2b + 1;
    n2bb = 0;
    ND1 = ND1 + 1;
    NA2 = NA2 + 1;
    if n1 > 0
        tD1 = t1a + random(truncateID);
    else
        tD1 = inf;
    end
    tD2b = t22a + random(truncateBag);
    D1(ND1, 1) = t1a;
    A2(NA2, 1) = t22a;
    event_counter1 = event_counter1 + 1;
    event_counter2 = event_counter2 + 1;
    ns1(event_counter1, 1) = n1;
    ns2(event_counter2, 1) = (n2a+n2b+n2c);
    times1(event_counter1, 1) = t1a;
    times2(event_counter2, 1) = t22a;
```

FIGURE 13



```
elseif (n2aa == 1 && n2bb == 1 && n2c == 0)
    t1a = tD1;
    t23a = tD1;
    n2c = n2c + 1;
    ND1 = ND1 + 1;
    NA2 = NA2 + 1;
    if n1 > 0
        tD1 = t1a + random(truncateID);
    else
        tD1 = inf;
    end
    tD2c = t23a + random(truncateBag);
    D1(ND1, 1) = t1a;
    A2(NA2, 1) = t23a;
    event_counter1 = event_counter1 + 1;
    event_counter2 = event_counter2 + 1;
    ns1(event_counter1, 1) = n1;
    ns2(event_counter2, 1) = (n2a+n2b+n2c);
    times1(event_counter1, 1) = t1a;
    times2(event_counter2, 1) = t23a;
end
```

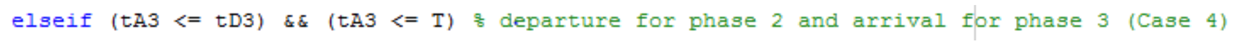
FIGURE 14



```
n2aa = 1;
n2bb = 1;
if tD1 < 10^307
    tA2 = tD1;
end
if min([tD2a tD2b tD2c]) < 10^307
    tA3 = min([tD2a tD2b tD2c]);
end
```

FIGURE 15

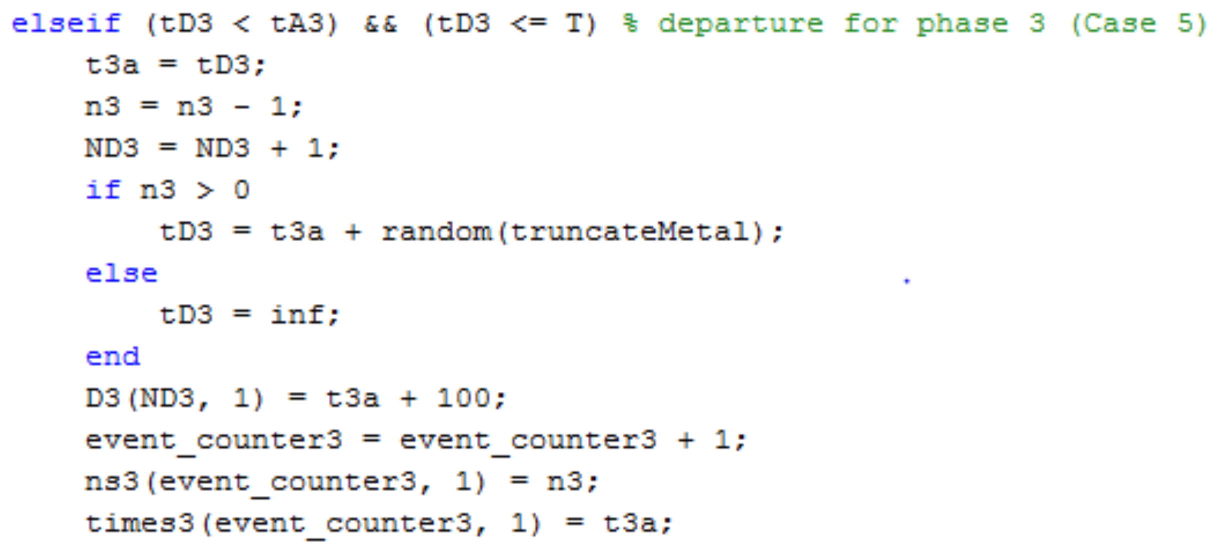
The third case is when a person in phase one is ready to depart and at least one of the lines in phase two is open and the time is less than the simulation time. Since someone has just left phase one, we subtract a person from phase one. The first instance of the third case is when the first line is open. Someone is then added to the first line and a departure time for that person is calculated. If someone is still in phase one after the departure, then a departure time for that person is calculated. If no one is in phase one after the departure, then the departure time for phase one is set to infinity because no one is there to depart. A placeholder signifying someone just arrived at the first line is then set to zero so no one else is added to other lines for this iteration of the case. The second instance of the third case is when the first line is not open and the second line is open. Someone is then added to the second line and a departure time for that person is calculated. If someone is still in phase one after the departure, then a departure time for that person is calculated. If no one is in phase one after the departure, then the departure time for phase one is set to infinity because no one is there to depart. A placeholder signifying someone just arrived at the second line is then set to zero so no one else is added to other lines for this iteration of the case. The third instance of the third case is when the first line is not open, the second line is not open, and the third line is open. Someone is then added to the third line and a departure time for that person is calculated. If someone is still in phase one after the departure, then a departure time for that person is calculated. If no one is in phase one after the departure, then the departure time for phase one is set to infinity because no one is there to depart. The placeholders are then set back to one so they can signify someone just arrived at either the first line or the second line for the next iteration of the case. The arrival time for phase two is then set to the departure time of phase one and the arrival time for phase three is set to the minimum of the departure times of phase two.



```
elseif (tA3 <= tD3) && (tA3 <= T) % departure for phase 2 and arrival for phase 3 (Case 4)
```

FIGURE 16

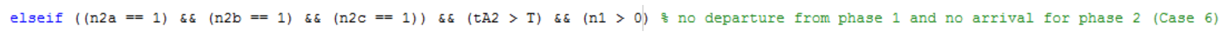
The fourth case is when someone is departing phase two and the time is less than the simulation time. This case follows the same guidelines as phase two except it does not include a departure from phase one. Therefore, all the methods are the same except a new departure time for phase one and arrival time for phase two is not set.



```
elseif (tD3 < tA3) && (tD3 <= T) % departure for phase 3 (Case 5)
    t3a = tD3;
    n3 = n3 - 1;
    ND3 = ND3 + 1;
    if n3 > 0
        tD3 = t3a + random(truncateMetal);
    else
        tD3 = inf;
    end
    D3(ND3, 1) = t3a + 100;
    event_counter3 = event_counter3 + 1;
    ns3(event_counter3, 1) = n3;
    times3(event_counter3, 1) = t3a;
```

FIGURE 17

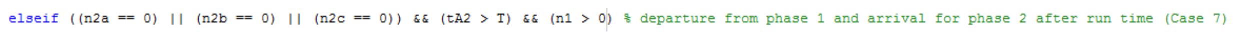
The fifth case is when someone is departing phase three and the time is less than the simulation time. Since someone has just left phase three, we subtract a person from phase three. If there is still someone in phase three, then a new departure time is calculated for that person. If there is no one in phase three, then the departure time is set to infinity since no one is there to depart.



```
elseif ((n2a == 1) && (n2b == 1) && (n2c == 1)) && (tA2 > T) && (n1 > 0) % no departure from phase 1 and no arrival for phase 2 (Case 6)
```

FIGURE 18

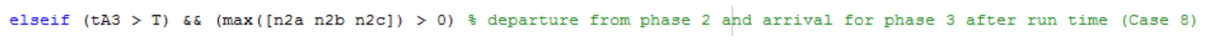
The sixth case is when a person in phase one is ready to depart and the time is greater than the simulation time, but all the lines in phase two are at capacity so they must wait. This case follows the same guidelines as phase two except it assumes there is someone in phase one, so any instance where there is no one in phase one is removed.



```
elseif ((n2a == 0) || (n2b == 0) || (n2c == 0)) && (tA2 > T) && (n1 > 0) % departure from phase 1 and arrival for phase 2 after run time (Case 7)
```

FIGURE 19

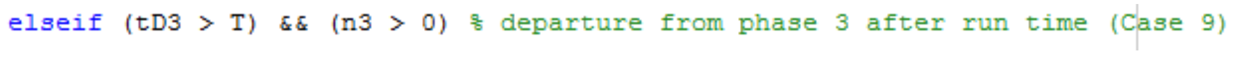
The seventh case is when a person in phase one is ready to depart and at least one of the lines in phase two is open and the time is greater than the simulation time. This case follows the same guidelines as phase three except it assumes there is someone in phase one, so any instance where there is no one in phase one is removed.



```
elseif (tA3 > T) && (max([n2a n2b n2c]) > 0) % departure from phase 2 and arrival for phase 3 after run time (Case 8)
```

FIGURE 20

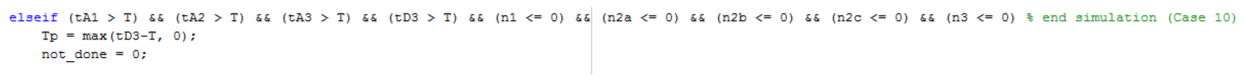
The eighth case is when someone is departing phase two and the time is greater than the simulation time. This case follows the same guidelines as phase four except it assumes there is someone in phase two, so any instance where there is no one in phase two is removed.



```
elseif (tD3 > T) && (n3 > 0) % departure from phase 3 after run time (Case 9)
```

FIGURE 21

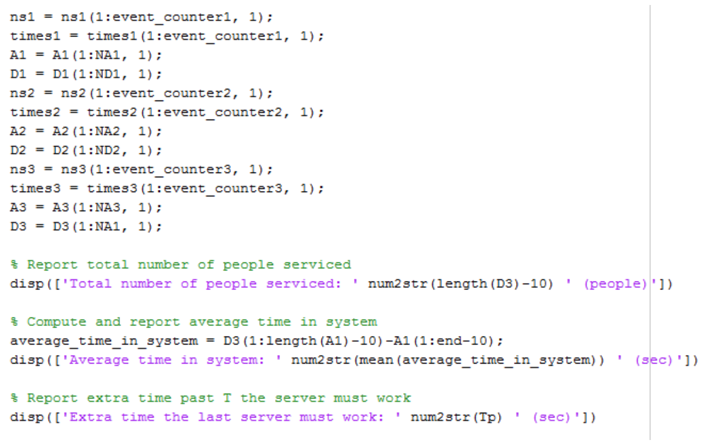
The ninth case is when someone is departing phase three and the time is greater than the simulation time. This case follows the same guidelines as phase five except it assumes there is someone in phase three, so any instance where there is no one in phase three is removed.



```
elseif (tA1 > T) && (tA2 > T) && (tA3 > T) && (tD3 > T) && (n1 <= 0) && (n2a <= 0) && (n2b <= 0) && (n2c <= 0) && (n3 <= 0) % end simulation (Case 10)
    Tp = max(tD3-T, 0);
    not_done = 0;
```

FIGURE 22

The tenth case is when no one is in any of the phases and the time is greater than the simulation time. A variable for time over the simulation time is calculated and the simulation is ended by setting a placeholder for the simulation to zero.

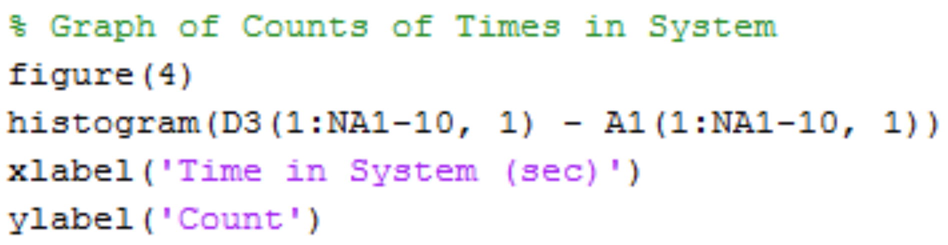


```
ns1 = ns1(1:event_counter1, 1);
times1 = times1(1:event_counter1, 1);
A1 = A1(1:NA1, 1);
D1 = D1(1:ND1, 1);
ns2 = ns2(1:event_counter2, 1);
times2 = times2(1:event_counter2, 1);
A2 = A2(1:NA2, 1);
D2 = D2(1:ND2, 1);
ns3 = ns3(1:event_counter3, 1);
times3 = times3(1:event_counter3, 1);
A3 = A3(1:NA3, 1);
D3 = D3(1:NA1, 1);

% Report total number of people serviced
disp(['Total number of people serviced: ' num2str(length(D3)-10) ' (people)'])

% Compute and report average time in system
average_time_in_system = D3(1:length(A1)-10)-A1(1:end-10);
disp(['Average time in system: ' num2str(mean(average_time_in_system)) ' (sec)'])

% Report extra time past T the server must work
disp(['Extra time the last server must work: ' num2str(Tp) ' (sec)'])
```

FIGURE 23

After the simulation is over, all the arrival times and departure times are stored in arrays as well as the amount of people in each phase during times of arrivals and departures. Total number of people serviced during the simulation time is then reported.

The average time in system for a person is then computed and reported. The extra time the last server must work after the simulation time is also reported.



```
% Graph of Counts of Times in System
figure(4)
histogram(D3(1:NA1-10, 1) - A1(1:NA1-10, 1))
xlabel('Time in System (sec)')
ylabel('Count')
```

FIGURE 24

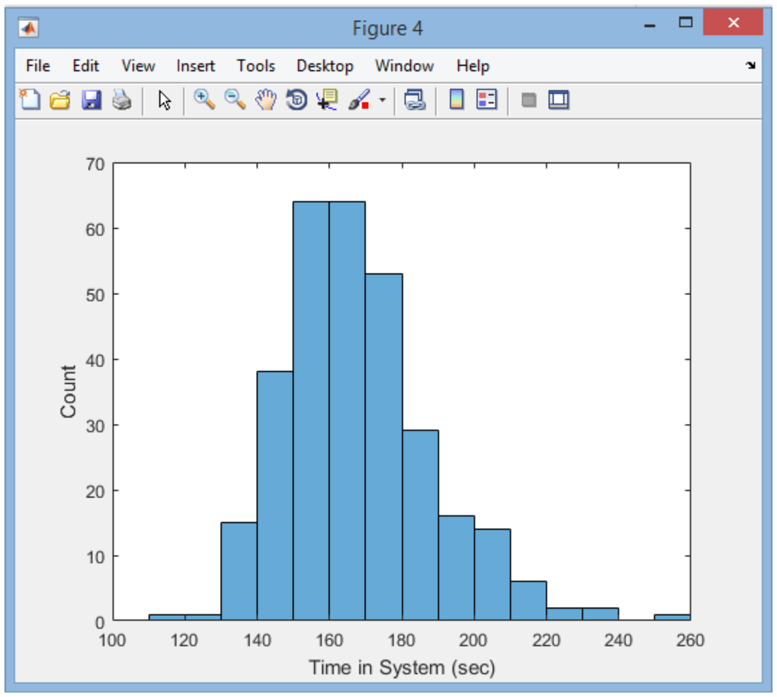


FIGURE 25 - TIME IN SYSTEM HISTOGRAM

A graph of times in system is created by subtracting phase one arrival times from phase three departure times. The graph is created as histogram showing the number of people who experienced different times in system.

```
% Graph of Time in System for Travelers
figure(3)
plot(D3(1:end-10, 1), average_time_in_system)
xlabel('Traveler Number')
ylabel('Time in System')
grid
```

FIGURE 26

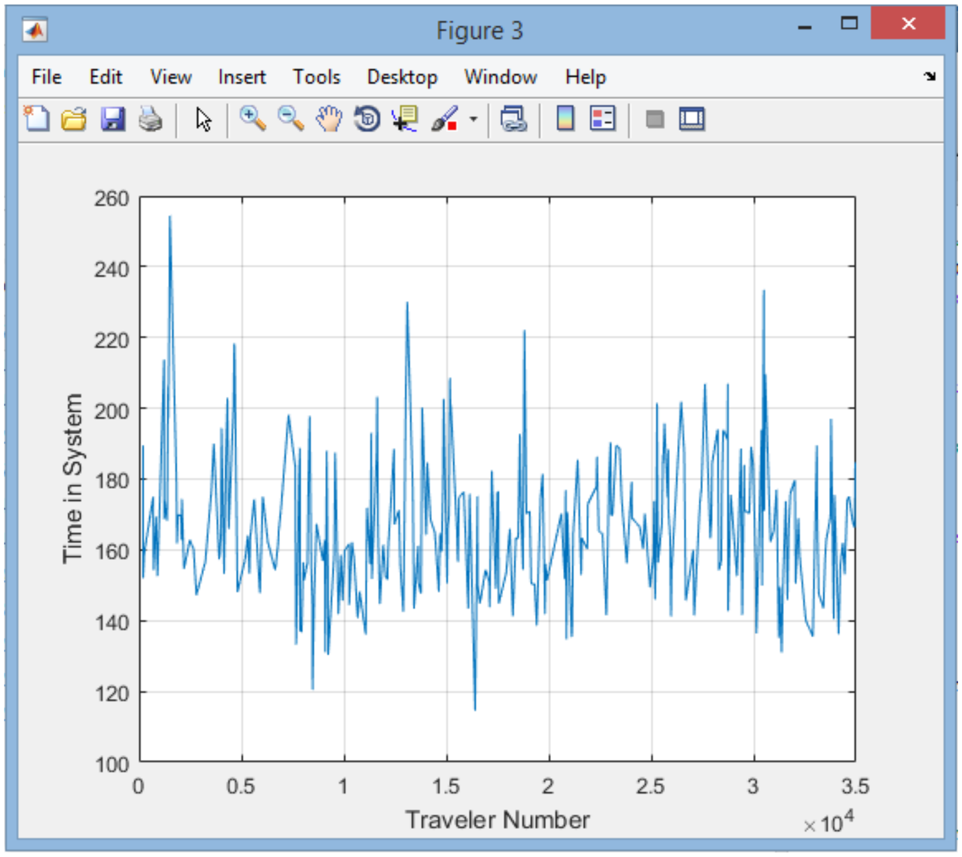


FIGURE 27 - TRAVELER TIME IN SYSTEM

A graph of time in system is created by displaying the average time in system during different times of the simulation. The graph is a line graph showing the average time in system during different times of the simulation.

```
% Graph of Total System Arrivals and Departures during Times
figure(2)
plot(A1, (1:NA1), 'r', D3, (1:NA1), 'b')
axis auto
xlabel('Time (sec)')
ylabel('Total System Arrivals (red) and Departures (blue)')
grid
```

FIGURE 28

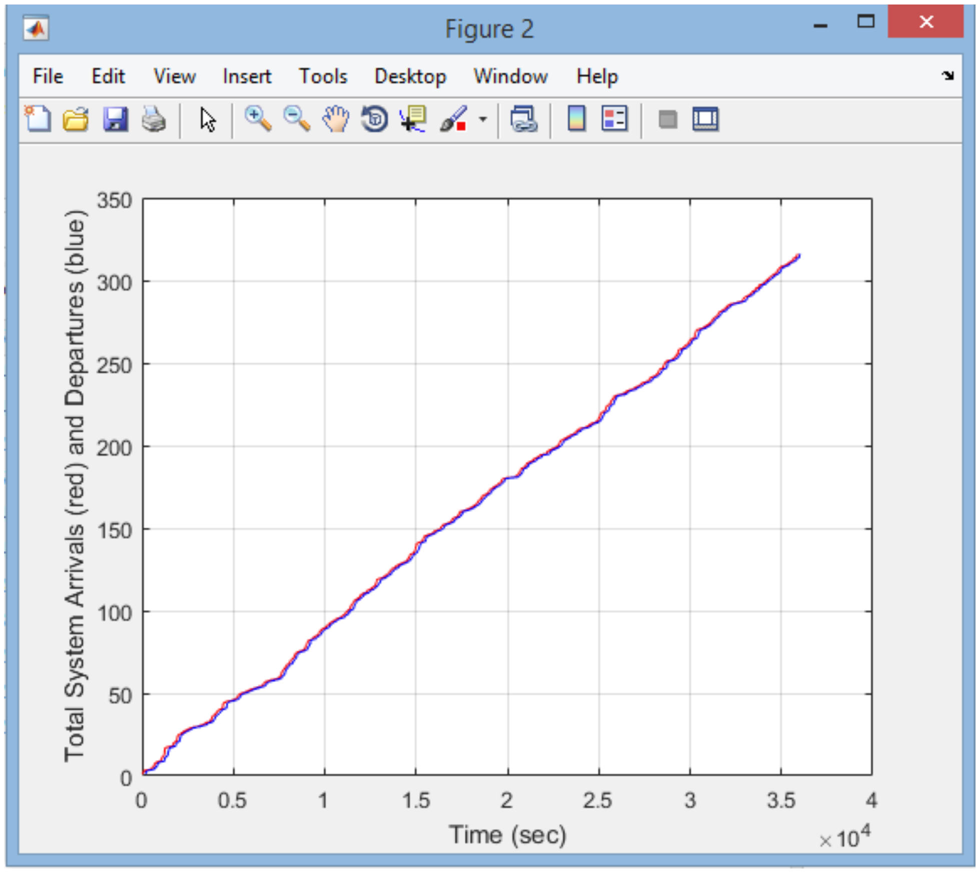
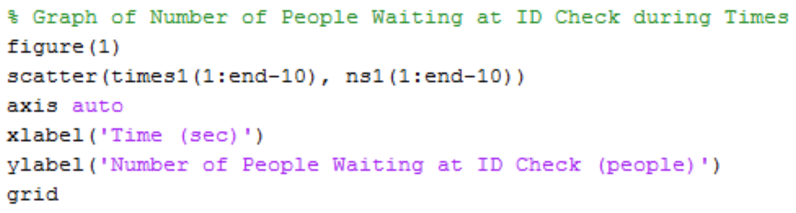


FIGURE 29 - TOTAL TIME SYSTEM ARRIVALS AND DEPARTURES

A graph of total system arrivals and departures is created by displaying all the arrivals from phase one and departures from phase three during their respective arrival and departure times. The graph is a line graph with two lines, one for arrivals and one for departures. If the lines are more spread out during a time, then the time spent in system is longer.



```
% Graph of Number of People Waiting at ID Check during Times
figure(1)
scatter(times1(1:end-10), ns1(1:end-10))
axis auto
xlabel('Time (sec)')
ylabel('Number of People Waiting at ID Check (people)')
grid
```

FIGURE 30

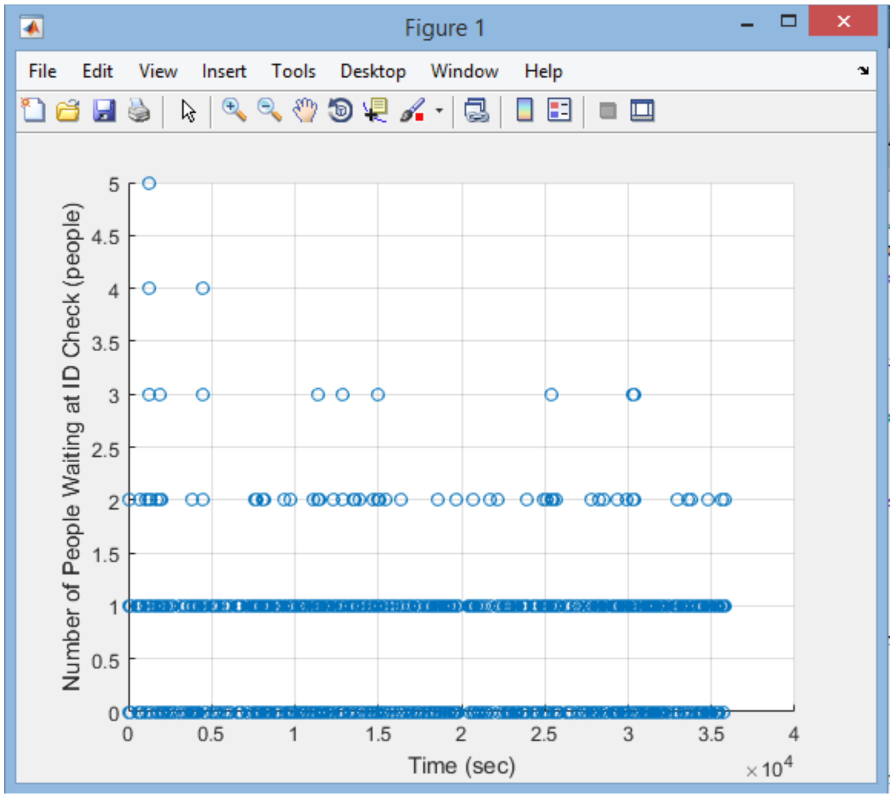


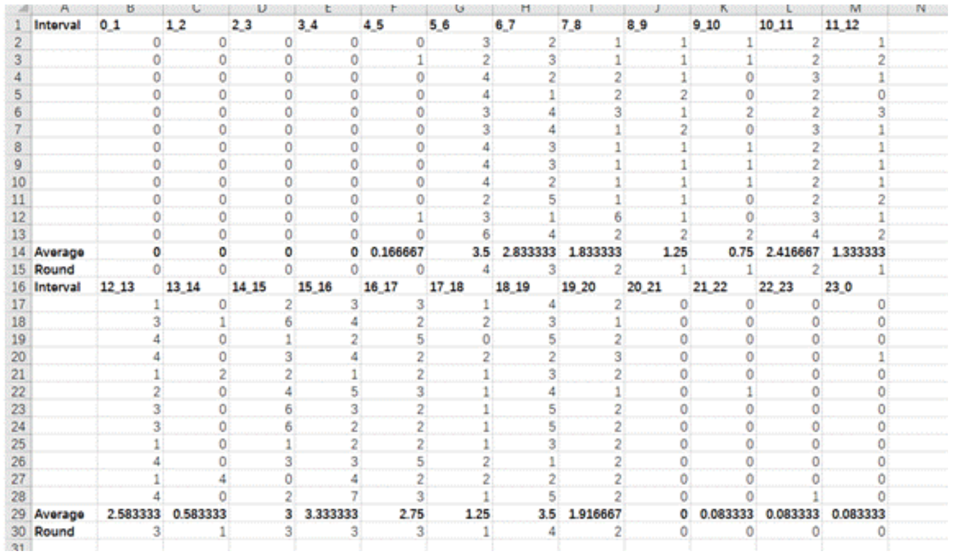
FIGURE 31 - TRAVELERS WAITING AT ID CHECK

A graph of waiting at ID check is created by displaying the number of people in phase one during times of arrivals and departures for phase one. The graph is a scatterplot showing the number of people in phase one during times of the simulation.

A final note is that the simulation is due to fail sometimes when it is run. Because it is a very complex simulation, memory leaks in hardware can cause the simulation to fail midway. However, most of the time the simulation runs perfectly. On a very powerful computer, the simulation should not fail at all. Please keep this in mind as the simulation is tested.

Analysis

After the simulation of security checkpoint, we decided to make use of this simulation by testing the system's performance in terms of different interarrival conditions. Since we assumed customers will plan their arrival to the airport according to the scheduled departure time, we'll only collect the scheduled departure time instead of the actual departure time, which will sometimes be delayed by weather conditions etc. According to the departure table of the CHO airport, we've come up with the statistics of departure frequency at different periods of 13 days.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	**Interval**	**0_1**	**1_2**	**2_3**	**3_4**	**4_5**	**5_6**	**6_7**	**7_8**	**8_9**	**9_10**	**10_11**	**11_12**	
2		0	0	0	0	0	3	2	1	1	1	2	1	
3		0	0	0	0	1	2	3	1	1	1	2	2	
4		0	0	0	0	0	4	2	2	1	0	3	1	
5		0	0	0	0	0	4	1	2	2	0	2	0	
6		0	0	0	0	0	3	4	3	1	2	2	3	
7		0	0	0	0	0	3	4	1	2	0	3	1	
8		0	0	0	0	0	4	3	1	1	1	2	1	
9		0	0	0	0	0	4	3	1	1	1	2	1	
10		0	0	0	0	0	4	2	1	1	1	2	1	
11		0	0	0	0	0	2	5	1	1	0	2	2	
12		0	0	0	0	1	3	1	6	1	0	3	1	
13		0	0	0	0	0	6	4	2	2	2	4	2	
14	**Average**	**0**	**0**	**0**	**0**	**0.166667**	**3.5**	**2.833333**	**1.833333**	**1.25**	**0.75**	**2.416667**	**1.333333**	
15	**Round**	0	0	0	0	0	4	3	2	1	1	2	1	
16	**Interval**	**12_13**	**13_14**	**14_15**	**15_16**	**16_17**	**17_18**	**18_19**	**19_20**	**20_21**	**21_22**	**22_23**	**23_0**	
17		1	0	2	3	3	1	4	2	0	0	0	0	
18		3	1	6	4	2	2	3	1	0	0	0	0	
19		4	0	1	2	5	0	5	2	0	0	0	0	
20		4	0	3	4	2	2	2	3	0	0	0	1	
21		1	2	2	1	2	1	3	2	0	0	0	0	
22		2	0	4	5	3	1	4	1	0	1	0	0	
23		3	0	6	3	2	1	5	2	0	0	0	0	
24		3	0	6	2	2	1	5	2	0	0	0	0	
25		1	0	1	2	2	1	3	2	0	0	0	0	
26		4	0	3	3	5	2	1	2	0	0	0	0	
27		1	4	0	4	2	2	2	2	0	0	0	0	
28		4	0	2	7	3	1	5	2	0	0	1	0	
29	**Average**	**2.583333**	**0.583333**	**3**	**3.333333**	**2.75**	**1.25**	**3.5**	**1.916667**	**0**	**0.083333**	**0.083333**	**0.083333**	
30	**Round**	3	1	3	3	3	1	4	2	0	0	0	0	
31														

FIGURE 32 - TABLE OF FLIGHT FREQUENCIES VS. PERIOD

In this table, we can read the expected frequency of flights at certain time periods. Our collection of interarrival data mostly comes from period 8-9, and at the days we collected our data, the flight frequency was always 1 flight/hour after the time we got to the airport. Since we assume most people will go to the security check roughly 1 hour before the departure, this simulation will provide the performance at the previous interval. For example, if the flight frequency is 1 at period 18-19, the result of our simulation will correspond to the airport at period 17-18. We also read that periods 0-4 and 20-0 are not significant, because either there are no flights departing at all, or there was only 1 flight coming on both of the days we collected. It is evident that at one time, the number of flights coming is either 0 or greater than/equal to 1, or discrete in nature. That's why we round the average flight frequency to the closest integer to get a more accurate simulation result. The default sample data for our simulation corresponds to the case where there is only 1 flight departure, and it is also the case for the minimum arrival rate when there is a flight departing.

Before actually testing the system, there is one point we have to address. In our simulation, we care most about the average time customers spent in the system and the distribution of their time spent in the system, and the exact number of customers arriving and departing is not necessarily important. Also, to make sure the simulation will produce a steady state expected value for average time spent in the system, we perform the Monte Carlo simulation step by simulating the system for multiple hours instead of simulating a 1-hour period. The reason why we do this is that if we only simulate a 1-hour period, the warm-up time will be equally as long as the simulation itself. Instead, with simulating a 10-hour period and setting a 1-hour warm-up time, we can get a steady state expected value from the simulation without having to run the code for a long time. In fact, if we simulate over a 100-hour period with warming up time equals to 10 hour, we are going to get a similar average time in the system. The following are our outputs with respect to different simulation times and warming up periods.

	# Hours of Simulation Time	**# Hours of Warmup Time**	**Total # of People Serviced**	**Avg. Time in System (sec)**	**Extra Time the Server Must Work (sec) Inf = no time**
Simulation 1	1	0.5	18	164	Inf
Simulation 2	10	1	317	165	Inf
Simulation 3	100	10	3113	168	36

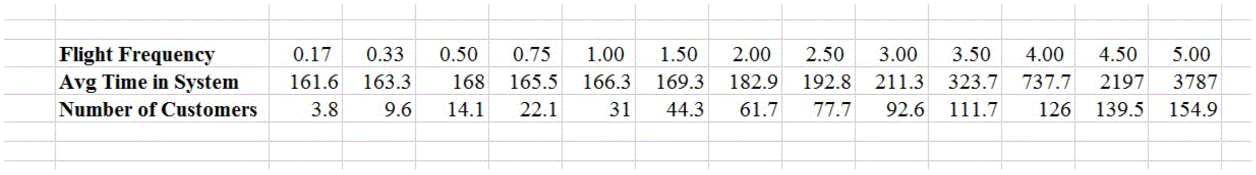
TABLE 1 - SIMULATION STATISTICS

It turns out the expected time in the system from simulating 1 hour is not that off, but the total number of people served is not directly proportional to the simulating time in this case (because 18 people for 1 hour, 317 for 10 hours and 3113 for 100 hours). We think the reason is a 1-hour simulation is not enough to get a steady state estimation. That is why we think it is optimal to simulate over 10 hours and get the amortized data for a 1-hour interval from calculation. In this case, if we use the sample data we collected and flight frequency = 1, the expected time in system will be roughly 165 seconds and the total number of people served will be about 310.

Experiment

After setting up the necessary simulation time for a steady state output, we would like to test the system's performance in different interarrival rates. Since we agreed the flight frequency should be discrete, we'll assume there exists a direct linear relationship between flight frequency and the arrival rate for the simplicity of our calculation. In real cases, this might not always be correct because different flights will have different number of customer capacity, and we don't have enough data to account for this difference. However, our simulation still provides insight to the system's performance in different arrival rates.

For analytical purpose, we also simulated the system's performance for interarrivals slower than the case where flight frequency is 1. We tested different interarrival conditions for 5 times and put the average time per customer spent in the system and average total number of customers together. The following are our results:



Flight Frequency	0.17	0.33	0.50	0.75	1.00	1.50	2.00	2.50	3.00	3.50	4.00	4.50	5.00
Avg Time in System	161.6	163.3	168	165.5	166.3	169.3	182.9	192.8	211.3	323.7	737.7	2197	3787
Number of Customers	3.8	9.6	14.1	22.1	31	44.3	61.7	77.7	92.6	111.7	126	139.5	154.9

FIGURE 32 - TABLE OF FLIGHT FREQUENCY AND CUSTOMERS

As we can roughly see, any flight frequency less than 1 will only change the number of customers (since less customers are coming to the airport), not the average time in system. So we can conclude that 165 seconds for every customer is the best performance the system is capable of. And if we accelerate the arrival of customers, the average time will expectedly increase. We can tell from the following graphs that when the flight frequency exceeds 3, the system will perform drastically worse. Below that level, the system will be slower gradually with respect to more frequent arrival. So we'll define the system to be at "regular" state if flight frequency is less than or equal to 3, and above 3, the system is "busy".

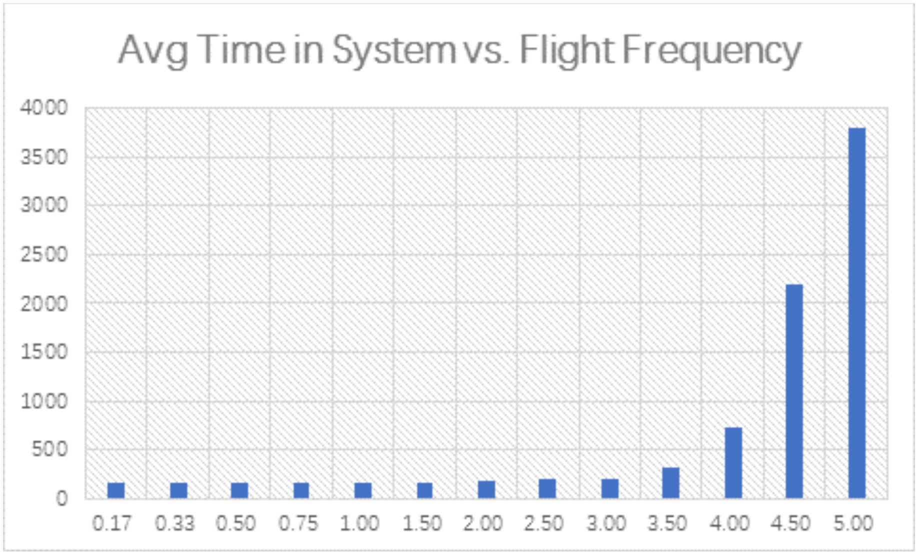


FIGURE 33 - AVG. TIME IN SYSTEM VS. FLIGHT FREQUENCY

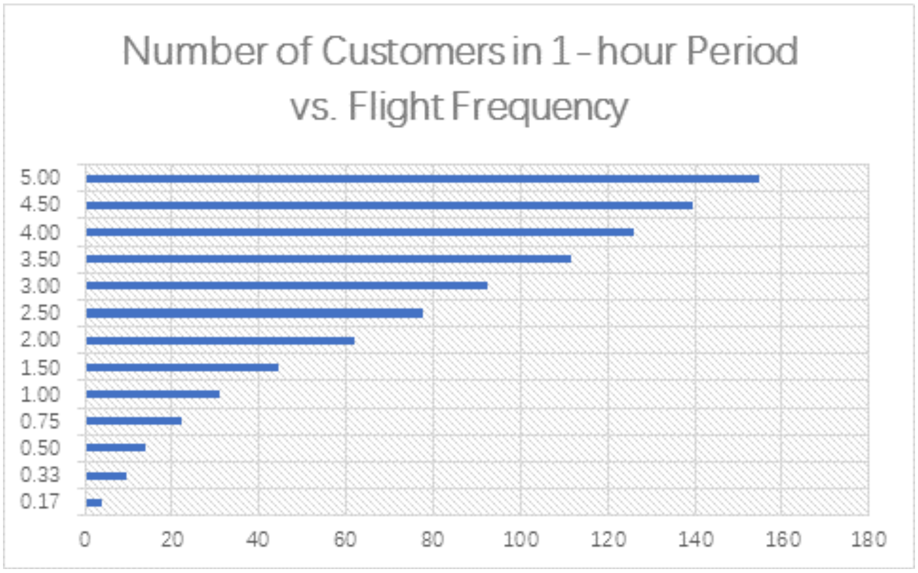
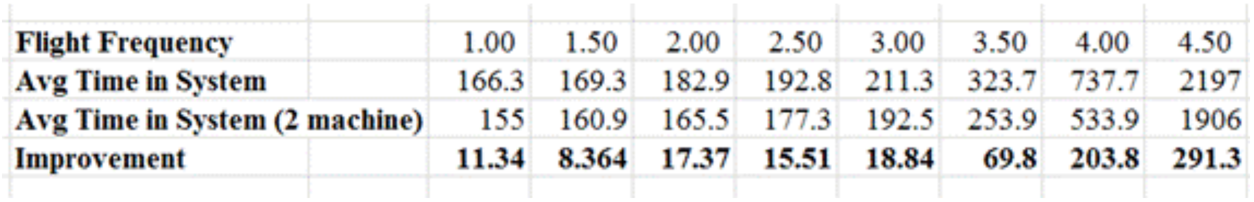


FIGURE 34 - NUMBER OF CUSTOMERS IN 1 HOUR PERIOD VS FLIGHT FREQUENCY

According to our observation at the airport, there are 2 machines for baggage check, and the staff will open the second one if they think it's necessary. We don't know how the staff define how "busy" the queue system is, but we'll try to find the optimal timing for opening up a second line for the baggage check because opening up a second machine requires more staff and energy, we have to make sure doing is worth it.

Due to the complexness of the simulation, here we will assume that having 2 machines for baggage check at the same time accelerates the service time for that process by halving the service time (changing service time for baggage check from 27.6 seconds to 13.8 seconds). We generated the result for several interarrival conditions with halved baggage check time:



Flight Frequency	1.00	1.50	2.00	2.50	3.00	3.50	4.00	4.50
Avg Time in System	166.3	169.3	182.9	192.8	211.3	323.7	737.7	2197
Avg Time in System (2 machine)	155	160.9	165.5	177.3	192.5	253.9	533.9	1906
Improvement	**11.34**	**8.364**	**17.37**	**15.51**	**18.84**	**69.8**	**203.8**	**291.3**

FIGURE 35 - FLIGHT FREQUENCY TABLE

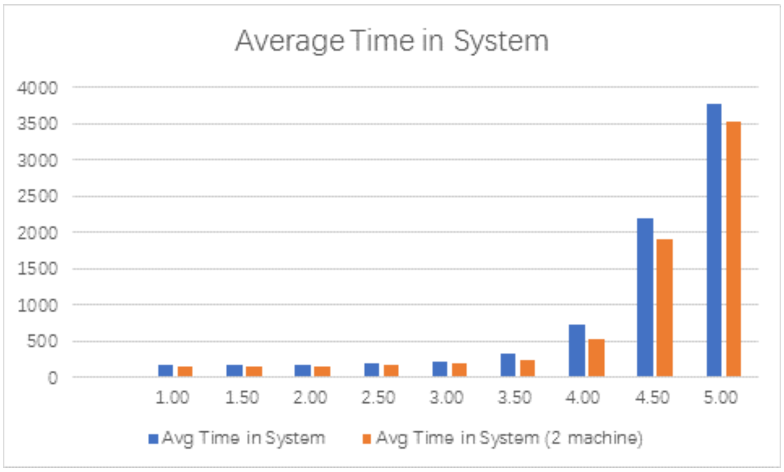


FIGURE 36 - AVERAGE TIME IN SYSTEM

As we can read from the results, at flight frequency below 3, the change of performance is not obvious because the improvement comes mainly from us cutting down the service time of baggage check, so there are actually no improvements to the performance. However, starting from flight frequency 3.5, which is the “busy” state of the system, we have a significant improvement in system performance, because the improvement is 69.8 seconds, which exceeds the improvement of the baggage check process (13.8 seconds). Then we conclude that when the flight frequency exceeds 3, it is necessary to always keep both baggage check machines working.

Recommendation

Besides opening up a second machine for baggage inspection, we also have more recommendations to the airport if they are considering improving this security checkpoint’s performance:

1. Training for staff to accelerate each process.
2. Recruit more staff at the checkpoints.

3. Encourage customers to use TSA Pre-Check so they have to spend less time at the baggage belts.
4. Encourage customers to check their baggage instead of carrying them on the plane.
5. Develop multiple ways of security check for people in need with special accommodations (disabled, aged etc).

Pitfalls of our Simulation

Our simulation is a simplified model of the CHO airport, so the deviations from the real system come from the assumptions we made to simplify this system. Here are some major sources of error:

1. In reality, people do not strictly follow an exponential interarrival model, because people tend to come to the airport in groups and check in together. Compared to group arrivals, individual tours happen not that often. However, we could not find an empirical model that is both simple to use and applicable for this situation. This difference will actually make the average time spent in system longer, because in terms of group arrivals, there exists a larger possibility that customers arrive at the same time and get stagnated at the entrance.
2. In reality, different customers will bring a different number of bags to the airport. So the baggage check will have different service times for everyone. Although we used the sample mean for all conditions, it is still worth mentioning that when a customer brings a lot of carry-ons to the airport, he/she should expect a longer time spent in the security check system.

3. In our simulation, we did not account for the cases in which the customers (or their bags) fail to pass the security check and they have to do the check again. Although we did not observe many of these cases happening in the airport, they are still possible and will give a longer service time for customers.
4. In our simulation, and the end of the loops we added an error term to account for the difference between our outputs and the sample mean of time people spent in the system. We think the main part of this error term is the time people spent between the checking processes, because in reality, people do not move uniformly in the system like machines. However, there is no way for us to conclude if this error term should be a constant or not due to the lack of data and complexity of this case. If we had more time, we could explore more about this error term and make the simulation more accurate.
5. There isn't an exactly linear relationship between the number of flights in an hour and number of travelers for a variety of reasons we cannot address easily. However, we believe the simplicity of the linear relationship still captures the nature of the system well.

We also made two MatLab functions for customers and staff. For_customer.m tells the customer if the security checkpoint will be busy 1 hour prior to their flights' departure and for_staff.m tells the staff if it is necessary to open up the second baggage check machine.

References

I. Janer, M. L., & Rosettie, M. D. (2016). Simulation Modeling of Alternatives to Avoid Interruptions of the X-Ray Screening Operation at Security Checkpoints, 0-End. Retrieved May 6, 2017, from http://www.informs-sim.org/wsc16papers/265.pdf

II. Scholtes, J. (2015, August 17). Price for TSA's failed body scanners: $160 million. Retrieved May 9, 2017, from http://www.politico.com/story/2015/08/airport-security-price-for-tsa-failed-body-scanners-160-million-121385