ANALYSIS WITH DYNAMIC BAYESIAN NETWORKS COMPARED TO SIMULATION

THESIS

Aaron J. Salazar, Captain, USAF

AFIT-ENS-MS-20-M-168

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENS-MS-20-M-168

ANALYSIS WITH DYNAMIC BAYESIAN NETWORKS COMPARED TO MONTE CARLO SIMULATIONS

THESIS

Presented to the Faculty

Department of Operational Sciences

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Operations Research

Aaron J. Salazar, B.S.

Captain, USAF

March 6, 2020

AFIT-ENS-MS-20-M-168

ANALYSIS WITH DYNAMIC BAYESIAN NETWORKS COMPARED TO MONTE CARLO SIMULATIONS

THESIS

Aaron J. Salazar, B.S.
Captain, USAF

Committee Membership:

Dr. Mark A. Gallagher, PhD
Chair

Dr. Lance E. Champagne, PhD
Member

Lt Col Eric L. Brooks, PhD
Member

AFIT-ENS-MS-20-M-168

Abstract

The Air Force Analyses and Assessment Directorate (AF/A9) is developing the Bayesian Enterprise Analysis Model (BEAM), which is a process to answer defense strategy and force structure questions. BEAM is a new type of military modeling with more aggregated resolution than campaign models. Demonstrating when, if ever, Bayesian Networks are a better or a comparable substitute to Monte Carlo simulation would assist in BEAM's development and implementation. A major underlying assumption of BEAM is that Bayesian Networks can produce a distribution of outcomes similar to the distribution from multiple replications of Monte Carlo simulations.

This research compares simulations to Dynamic Bayesian Networks in analyzing situations. The research applies models that have known output mean and variance. Queueing systems have theoretical values of the steady-state mean and variance for the number of entities in the system. Monte Carlo simulation development is broken down into two separate approaches: discrete-event simulation and time-oriented simulation. The discrete-event simulation is an object-oriented modeling approach where model progression is triggered from using pseudo-random numbers to schedule future events. The time-oriented simulation utilizes fixed-width time intervals and updates the system state according to a stochastic process for the set of events occurring during each time period. The accuracy of each approach is estimated by a comparison to the theoretical mean, variance, and state probability values. The Bayesian approach does produce state space probabilities very similar to the Monte Carlo simulations approaches.

Acknowledgements

I would first like to thank my advisor, Dr. Mark Gallagher, for his expertise, guidance, and encouragement throughout my time at AFIT. I also thank my committee members, Dr. Lance Champagne and Lt Col Eric Brooks, for their support and input for this research. Additionally, I thank Maj Thomas Talafuse for his additional suggestions and contributions. Finally, I would like to thank my friends and AFIT classmates for their support.

Aaron J. Salazar

Table of Contents

Page

List of Figures

List of Tables

ANALYSIS WITH DYNAMIC BAYESIAN NETWORKS COMPARED TO MONTE CARLO SIMULATIONS

I. Introduction

1.1 Background

The continual development and augmentation of today's technologies often provide new insights as well as improved effectiveness for any given task. Operations Research (OR) is the application of scientific and mathematical methods to the study and analysis of problems involving complex systems. Along with optimization and applied mathematics, simulation is often a standard method of analyzing data that deals with risk and uncertainty. While useful, Monte Carlo simulations may sometimes come at a trade-off between, accuracy, model run-time, and possible insights. In instances with large complex systems, simulations may have long run-times, requiring both replications and separate instances for each unique set of system parameters.

Advances in computation speed may allow Bayesian Networks to serve as a potential alternative to a Monte Carlo simulation. In an attempt to improve Bayesian Network computational speed, the Johns Hopkins University Applied Physics Lab (JHU/APL) is researching to reformat inputs of a Bayesian Network as a series of nodes compared to a traditional format that includes every permutation of influenced nodes in the system. Rather than computing all possible permutation of nodes simultaneously, the computations are handled two per iteration, containing a new input node in each step. This sequentialization significantly reduces the Bayesian Network computational time while producing the same results under certain conditions.

Additionally, JHU/APL researchers found that number of bins used in a Bayesian approach may be greatly reduced with minimal loss in accuracy [16] [15]. Rather than using fixed-width bins for probability distributions, they use constant-probability, which results in fewer required bins and thus further reducing computational time. Partitioning a distribution into fewer bins based on constant-probability allows for the reduction of computations spent on rare events. The constant-probability bins greatly reduces overall computation time while not substantially affect results.

The primary mission of the United States Department of Defense (DoD) is to provide military forces needed to deter war and to protect the security of the nation. Operating at a high operations tempo, the DoD requires expedited, as well as accurate, research to efficiently support the war fighting mission. To safeguard national security and to improve United States military capabilities, the DoD must continually progress and evaluate military strategies, tactics, and weapons systems. Currently, the predominant method for analyzing uncertain events and future predictions is simulation, or more specifically, combat modeling. Due to complexity and size of these models, implementation can often be time-consuming. A long simulation setup-time has the possibility to limit the ability to effectively evaluate the diverse strategy and structure trade-space under the vast plausible conflict types. Analysts need effective and efficient tools to evaluate the diverse possibilities of scenarios.

1.2 Problem Statement

The Air Force Analyses and Assessment Directorate (AF/A9) is developing the Bayesian Enterprise Analysis Model (BEAM), which is a process to answer defense strategy and force structure questions. BEAM is a new type of military modeling approach with more aggregated resolution than campaign models. BEAM provides different evaluation tools that the DoD can use to assess military strategies and de-

fense force structures across multiple campaign scenarios [8]. BEAM may provide useful insights to the Air Force and DoD senior leaders. Demonstrating when, if ever, Bayesian Networks are a better or comparable substitute to a Monte Carlo simulation would assist in BEAMs development and implementation. A major underlying assumption of BEAM is that Bayesian Networks can produce a distribution of outcomes similar to the distribution from multiple replications of Monte Carlo simulations.

1.3 Approach and Research Objectives

This research compares Monte Carlo simulation techniques to constructed Bayesian Networks in analyzing situations. It begins with models with known output mean and variance. Queueing systems have theoretical steady-state values for mean and variance of the number of entities in the system. Reliability models with any combination of series and parallel constructs are another application where we have a known theoretical mean and variance. The accuracy of each approach is compared by estimating the theoretical mean, variance, and probability values. Additionally, this examines the computational setup time and run-time of all approaches. The model results are then summarized in an attempt to seek insight into how the complexity of the underlying analytic situation impact performance and overall computation time.

Analysis associated with either Monte Carlo simulation or Bayesian Networks require specialized knowledge and training. Both of these techniques are considered to be highly specialized and require proper knowledge in order to extract proper insights or influences. An incorrectly constructed simulation or Bayesian Network is highly susceptible to bias or error. Simulation software requires highly-trained and knowledgeable subject matter experts (SMEs) to correctly model and interpret systems or interactions. Analytic software often is expensive and requires continued updates and support from the developers. This research does not investigate the

training or education requirements necessary to implement either approach.

1.4 Summary

Today's technologies are evolving at an extremely fast rate with abundant new techniques and theories. When dealing with uncertainty or unknown parameters, there exist a multitude of proven techniques to estimate the underlying distribution and behavior of a modeled system. A popular modeling technique used today is Discrete-Event Simulation. Time-oriented situations are applied when interaction between entities makes predicting the time of future simulated events difficult. The DoD relies heavily on simulation techniques and software to mimic actions for future situations. Due to trade-offs, the DoD is continually looking to improve ways of operating. Bayesian Networks, developed and applied correctly in certain settings, may provide an alternative to Monte Carlo simulations. Demonstrating that a Bayesian Network can serve as an adequate substitute to simulation can possibly help the further development of BEAM and most importantly, provide a new technique that the DoD can implement for analysis. This research compares Bayesian Networks to Monte Carlo simulations to explore the technical differences between the differing techniques on the bases of computation run-time and accuracy of the predicted outcomes.

II. Methodology

2.1 Overview

The primary focus of this research is to develop and compare different types of modeling techniques. In order to adequately assess each technique, a test case is utilized as a baseline approach that has known theoretical steady-state values. These known values are useful in determining the overall accuracy and validity of a model. Such a system that meets this criteria is an M/M/1 queue. More precisely, the state of interest is the length of the system for a single server with exponential inter-arrival and service rates. Three modeling techniques are tested and compared: Discrete-Event Simulation (DES), Time-Oriented Simulation (TOS), and a Dynamic Bayesian Network (DBN).

DES is a simulation technique which deals with events triggered by object generation within the system. TOS is similar to DES development but utilizes a user-defined time interval with likelihood of event occurrences. The time-oriented approach is useful in situations when the occurrence time of simulated events cannot be determined, such as when the interaction of entities affects the time of events. A military example is predicting when the radar from one aircraft will detect another aircraft. TOS is specifically useful in this research because it closely aligns with the development of the DBN model, as both techniques use the same system states and transition probabilities.

2.2 Queueing Theory

Queueing theory may be described as an analytical study of waiting lines for a specific system of interest. The queueing theory studies processes where the need to carry out certain tasks (i.e. services) arises on one side, and the need to fulfil them

arises (i.e. arrivals) on the other side of a system [6]. Models may vary in a number of parameters which includes entity inter-arrival times, service times, and amount of servers in the system. The movement of an entity through the system is an assumed instantaneous transfer between system states, thus not requiring additional travel time or a resource required for transition.

The queueing system of interest is defined with three primary parameters: inter-arrival rates, service rates, and the quantity of servers in the system. As previously mentioned, a commonly modeled system is an M/M/1 queue, which represents a system that follows a Markovian Poisson process. This system has a known exponential customer inter-arrival rate, exponential service rate, a single server, and no limit on queue length. Additionally, each server will have a specified capacity, which in this case is one system entity. The more general notation is an M/M/c queue, where c is the number of servers in the system. The exponential probability density function is derived as:

$$f(x, \lambda) = \begin{cases} \lambda e^{-\lambda x}, & \text{x} > 0 \\ 0, & \text{x} < 0, \end{cases} \tag{1}$$

where λ is the rate parameter of arrivals per time interval T and defined as the reciprocal of the mean.

Table 1 displays commonly tracked metrics in queueing models along with their theoretical steady-state calculations. L_S is the length of the system, including both the queue and server, while L_Q is the average number of customers in the queue. W_S represents the average amount of time that the customer spends in the entire system, while W_Q is the amount of time a customer waits for service while in the queue [14].

In the specific case of the M/M/1 queueing system, λ represents the customer inter-arrival rate, and μ is the rate parameter of service completions per time interval T for their respective exponential distributions. The parameter $\rho = \frac{\lambda}{\mu}$ is the theoret-

Table 1. M/M/1 Queue Theoretical Characteristics [14]

Parameter	Equation	Description
ρ	$\frac{\lambda}{\mu}$	Proportion of time server is busy
L_S	$\frac{\rho}{1-\rho}$	Length of the system
W_S	$\frac{1/\mu}{1-\rho}$	Wait in the system
L_Q	$\frac{\lambda^2}{\mu(\mu-\lambda)}$	Length of the queue
W_Q	$\frac{\lambda}{\mu(\mu-\lambda)}$	Wait in the queue

ical proportion of time that the server is busy in steady state. To properly achieve a stable system without balking or reneging, the system must have $\lambda < \mu$.

As ρ approaches the value of one, the system will have periods with more entities in the system. This system contains a single server queue with an infinite system capacity. The entities in the queue are processed as first-in-first-out (FIFO) service meaning that the system entities are serviced in the order in which they arrive. The system characteristics, including system length and entity waiting time, are determined solely based on the parameters λ and μ.

2.3 M/M/1 Queue

For a comparable application of the three modeling techniques, an M/M/1 queue will be used with its theoretical outcomes and state probabilities of the system. Specifically, each model estimates the average number of customers in the system (L_S). A practical example of this system is a bank with only one open service counter or a grocery store with a single self check-out kiosk. This system will utilize a user-defined λ and μ such that $\lambda < \mu$. As part of this analysis, these rate parameters will have varied combinations along with varied magnitudes. The magnitude affects each model's predictive capability as it may change the required time to reach steady-state, time interval size, or additional calculations due to a larger range of variability of possible state values. A primary concern is to ensure that each model executes for a duration

that properly allows the system to reach steady-state characteristics.

Additionally, each model must generate enough data points to properly analyze the system's behavior. Enough model entities or observations must occur to adequately mimic the system. For instance if $\lambda = 0.01$, the expected number of arrivals within the first 100 time units is one. In this hypothetical scenario, it is possible that even 1000 time units might not be of sufficient length to generate enough simulated arrivals to observe the system since the expected number of arrivals in this amount of time is only 10.

This single server system contains a server capacity of one entity (i.e. customer) and has instantaneous transfer time. The system does not allow for balking or reneging, defining it as a true birth-death Poisson process with the known exponential inter-arrival and service rates.

2.4 Discrete-Event Simulation (DES)

Simulations are the imitations of the operation of a real-world process or system over time [2]. Simulations are useful to draw inferences of system operation characteristics in various scenarios based on known or hypothesized parameters. A simulation could be accomplished manually, however time-consuming and difficult to track. However, computers, along with their current high capacity of computing, allow simulations to be executed in a more efficient manner at higher speeds when compared to manual methods. The development of the DES model involves defining discrete states, assuming that a system may not contain partial entities or events. Additionally, events occur at discrete moments in simulated time even though time is a continuous variable. The DES modeling technique is considered dynamic, discrete, and stochastic. It represents a system over time while containing input random variables defined in discrete states.

The event-based DES modeling approach schedules events at random. The random event generation follows a theorized or empirical distribution where a pseudo-random number is utilized. The algorithm used to generate the pseudo-random number varies between software applications. In this research, the Mersenne Twister random number generator is used to generate the pseudo-random number values. The Mersenne Twister algorithm is one of the most widely used random number generators, especially within high performance computing applications used by large financial institutions [19]. The algorithm sequence of a pseudo-random number generator is not truly random because it relies on an initial prime value. The Mersenne Twister algorithm's complexity allows for a possible $2^{19937}-1$ numbers prior to cycling; Tian and Benkrid [19] prove its pseudo-random numbers to be equally distributed in 623 dimensions. The algorithm's long cycle duration and fast computational time leads to its popularity among other generators.

The DES random events are rank-ordered on a timeline. The user (or system) can then parse to observe the system's behavior. Since a discrete-event model generates randomly scheduled events, either a very long replication or multiple replications are necessary to assess the average system parameters. The system may also have an initial transient prior to achieving steady-state operation. The simulation graph displayed in Figure 1 shows the system behavior of an M/M/1 queue. The system contains three primary nodes: Arrival, Service Begin, and Service End. The five characteristics of this system at a given time, t, includes the $Q(t)$ length of the queue and $S(t)$ status of the server. The next state transition is either determined by the next scheduled $a(t)$ arrival t_A arrival time or the next scheduled t_S service time.

To obtain a probability distribution of end states, the probability of system length, L_S, is estimated by percent of time the simulated system status was in that state.

Initialization bias may be present due to the beginning status of the modeled

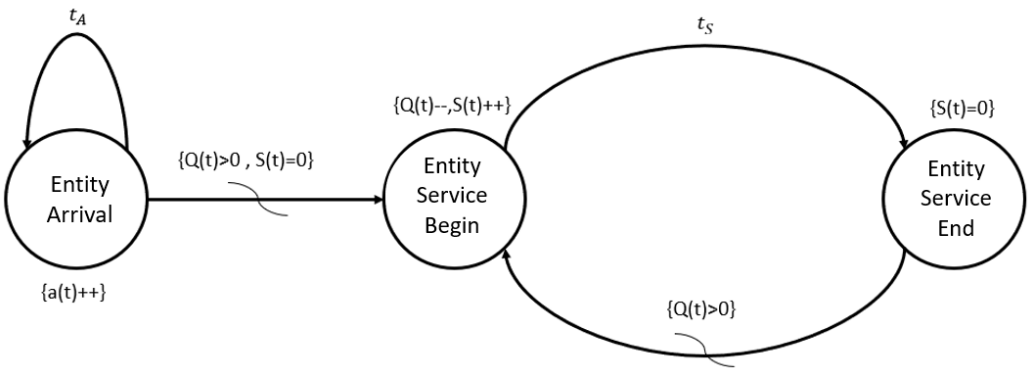

Figure 1. Simulation Graph: M/M/1 queue

system. For proper comparison, all modeling techniques will begin with a system that is empty and idle. Since the desired metric tracks L_S, model conditions must be considered to allow the system to reach steady-state characteristics. Conway states that there are also circumstances where steady-state measurements cannot be obtained due to reasonable sets of system parameters [4]. One such circumstance includes an unstable system when the mean service and arrival rates are equal. Rather than truncating initial warm up time, initialization bias is accounted for by ensuring that the model is executed for a long enough time such that the system reaches steady-state behavior without influence of the initial starting condition. The warm up period truncation may be somewhat arbitrary and differs between modeled system [21].

2.5 Time-Oriented Simulation (TOS)

Time-oriented simulation (TOS) is similar to DES, but defined as a time series progression rather than event scheduling for each system entity. This time series progression contains a fixed-width time interval in which the number of events are determined to occur within the specified time interval. This simulation approach is also referred to as fixed-increment time progression due to its nature of time interval

formulation. Buss and Rowaei develop similar discrete-time simulation models, concluding that the choice of time interval size has substantial impact on model results and execution time [3]. The TOS modeling technique utilizes static determinations of system states as it progresses over time. While this is considered dynamic due to its progression over time, states are determined within each static time interval and thus, also including a Monte Carlo approach. For the sake of comparison, the DES and TOS models are simply referred to as simulations, not including the Dynamic Bayesian Network.

The DES model's object-focused progression captures data and system characteristics at the point of the randomly generated events, while in contrast, TOS evaluates the system state at each individual fixed time interval. Both simulation techniques produce unique output, thus requiring either a single long replication or multiple replications of shorter length to capture the true system behavior. Time-oriented simulation takes an equivalent timeline and splits the time space into user-defined fixed intervals. The progression through the TOS model uses a stochastic process with theorized probabilities of event occurrences while updating system states according to the set of events or activities occurring within each time interval.

The TOS model uses a pseudo-random number to determine the amount of simulated arrivals and service completions within each time interval. Specifically, a random number is compared to a calculated cumulative probability distribution to determine the number of arrivals or service completions. The service completion distribution depends on the status of the system and amount of arrivals within a time interval.

Within simulation model progression, the system entities that exist at the end of the prior time interval plus new arrivals during the current time interval limit the amount of possible service completions. Since M/M/1 queue has exponential distributed service completions, if an infinite number of service completions were

theoretically possible, the number of service completions would follow a Poisson distribution. However, the service completions are limited due to logic which dictates that a service may not be performed for entities that do not yet exist in the simulated environment. The result is the possible service completions follows a truncated distribution, which is dependent on system status of number in the system at the beginning of time interval and the number of arrivals during the time interval. This research proposes and tests four different approaches to approximate the service estimation distribution for the TOS model.

The TOS and DBN model development requires an appropriate estimation of the system's service characteristics. For an M/M/1 queue, the defined exponential arrival rate λ along with a pseudo-random draw from the appropriate Poisson distribution, determines the amount simulated arrivals within each time interval. The service completions also follow this generation technique with rate of μ. The service completions, however, require approximations because their likelihood are dependent on the status of the system at time t and the number of arrivals. A stable queue requires that $\lambda < \mu$, meaning that the system has a higher rate of service than arrivals. The number of arrivals during a time interval may be modeled with a Poisson distribution; however, the Poisson distribution with probabilities that extend to infinity does not fit service completions because the system cannot service entities that are not yet present. The service probabilities are dependent on the system state, specifically system length (L_S) and amount of arrivals within a time interval (A). The service rate follows an exponential distribution, so the four approximations for the number of service completions are based on transformations of a Poisson distribution. The Poisson probability density function (pdf) is defined as:

$$Poisson(\mu) = \frac{\mu^x e^{-\mu}}{x!} \tag{2}$$

where μ is the rate.

A service probability distribution derivation should properly account for the state of the system which includes system length L_S and the number of arrivals A. The system length L_S, exists at the beginning of each time interval and thus, no adjustment is required for the service probabilities pertaining to states zero through $L_S - 1$. The arrivals during the interval, however, require separate consideration to account for randomness of the time of each arrival.

Any new arrival is unlikely to have occurred precisely at the beginning of the interval, so a time interval adjustment, equivalently the rate of service, is required to account for a shorter time the arrivals have to be serviced. Thus, the new arrivals should have a lower probability to be serviced because they have a shortened time in the system, which is dependent on their exact arrival time. However, in TOS, the actual arrival times are not determined. Since the inter-arrival times follow an exponential distribution, the spacing of arrivals in a time interval conditioned on the number of arrivals follows a uniform distribution [20]. (See [1] for proof.) Thus, the expected portion of remaining time t_j after the j^{th} of A arrivals in a time interval of duration T is:

$$t_j = \frac{(A - j + 1)}{A + 1}, \text{ for j = 1,..., A} \tag{3}$$

For instance, in the case $A = 2$, the first arrival will have the expected portion of time remaining $t_1 = \frac{2}{3}$, and the second arrival will have expected portion of time remaining $t_2 = \frac{1}{3}$. All of the L_S entities in the system at the start of the interval will have the entire time interval for service.

2.5.1 Service Probability Bounds

The service probability calculations vary depend upon the assumptions that affect the likelihood of a new arrival completing service within a time interval. States in

the service probability density function (pdf) range from zero up to the length of the system from the previous time interval plus new arrivals in the current interval ($L_S + A$). The simulated number of service completions follows a distribution where the service of entities in the system at the beginning of the interval generally follow a Poisson distribution, and the service of arriving entities require adjustments due to their decreased time in the system. Once the service probability distribution is determined, a pseudo-random draw is used to select the simulated number of service completions in the time interval. The TOS model progresses through time and defines system states as L_S, which is dependent on the number of arrivals and service completions occurring within each time interval. Service estimation bounds help identify accuracy while guiding the approach's development. Additionally, these bounds assist with debugging and identifying possible logic errors.

An assumption that all arrivals have the entire time interval length to be serviced constitutes a lower bound for the overall mean length of the system. This assumption follows that along with the system length (L_S) from the prior interval, the new arrivals appear precisely at the beginning of the interval. Since all entities exist at the beginning of a time interval, the service rate or probabilities are not adjusted to account for simulated arrival times during the interval. Thus, μ serves as the exponential service rate that derives the service probability distribution for all entities. The assumption of arrivals occurring at the start of the interval serves as a lower bound for the average length of the system because it allows the entire interval to service the existing system length plus new arrivals. The resulting calculated mean for number in the system is lower than the theoretical steady-state L_S value because the calculation does not account for arrivals occurring too late to complete service.

Similar to that of the lower bound of average length of the system, an upper bound may also be obtained with an assumption that all arrivals during an interval occur

at the end of the interval. This assumption defines arrival time such that entities have no chance to complete service within the time interval in which they arrive. Thus, service completions are only considered for the existing system length, while disregarding opportunities for the service completion of new arrivals. In this case, the calculated mean of the length of the system is higher than the theoretical steady-state value because the model services fewer entities than technically possible.

The following subsections describe four approaches to estimate the service completion distribution given the prior number in the system and the number of arrivals, which is determined by a pseudo-random draw.

For each of the four approaches, L_S and A are predetermined values prior to each time interval. The random variable Z_j represents the number of entities serviced within time interval j. Z_j is conditioned on the predetermined L_S and A values at the beginning of the interval. A pseudo-random draw determines the service amount after all service probabilities are calculated. The service amount is then used to calculate S_j, representing the amount of entities remaining at the end of time interval j. The new system length for the next time interval is equal to the result of the current state, so $L_S = S_j$. Since the value $L_S + A$ truncates the service probability table, there exists a residual probability from the right tail of the Poisson distribution that is not feasible due to system status. This approach aggregates this residual probability value into state L_S. If there exists a probability of servicing more than L_S, then servicing L_S should have a higher likelihood of occurrence, and thus absorbing the remaining probability to ensure that the discrete probability density function (pdf) sums to one. This residual probability assumption will be discussed in more detail later in this chapter.

2.5.2 Service Estimation - Approach #1: Current State, Conditioned on Prior Entities

This first approach focuses on appropriately adjusting service rate μ based on the assumed uniform arrival time where t_j estimates the portion of time remaining for the j^{th} arrival. This approach models arrival services based on the system assumption of a first-in-first-out (FIFO) system. An arrival may only be serviced if all prior entities in the system (L_S) and prior arrivals are serviced. The j^{th} arrival's service is dependent upon the expected portion of remaining time in the interval. The random variable X is defined for the entities in the system at the beginning of the time interval (L_S), and $X_j^{'}$ for entities arriving during the interval, each following a Poisson distribution such that:

$$\begin{aligned} \mathrm{X} &\sim Poisson(\mu) \\ X_j^{'} &\sim Poisson(\mu t_j) \end{aligned} \tag{4}$$

The random variable Z_i is the distribution of service completion during the i^{th} time interval. The number entities in the system prior, L_S, and the number of arrivals A are known prior to determining the service completions Z_i.

Service Estimation Approach #1:

$$\mathrm{Z}_i = \begin{cases} \mathrm{Prob}(Y_i = j) = \mathrm{Prob}(X = j) & \text{for } j = 0, ..., L_S - 1 \\ \mathrm{Prob}(Y_i = j) = 1 - \sum_{k=0, k \neq L_S}^{k=L_S+A} \mathrm{Prob}(Y_i = k) & \text{for } j = L_S \\ \mathrm{Prob}(Y_i = j) = \mathrm{Prob}(Y_i = j-1)\mathrm{Prob}(X_{j-L_s}^{'} = 1) & \text{for } j = L_S + 1, ..., L_S + A \end{cases} \tag{5}$$

for $i = 0, ..., L_S + A$

The random variable for number of entities in the system, S_j, at the end of interval

may be described by equation 6.

$$S_j = L_S + A - Z_j \text{ for } j = 0, ..., L_S + A \tag{6}$$

In TOS, a pseudo-random draw is used along with the distribution for Z_i to determine the number of service completions in the interval and the number in the system at the end of the time interval.

2.5.3 Service Estimation - Approach #2: Scaled Probabilities

The second service estimation approach also focuses on new arrival service probability calculations. This approach calculates a service probability evaluated from the Poisson distribution and multiplies the value by a scalar to account for the varying time of arrivals within a time interval. For the entities arriving within an interval T, this approach scales the probabilities based on the expected total time that arrivals are present in the system. Again, the random variable X follows a Poisson distribution with rate μ.

Service Estimation Approach #2:

$$\mathrm{Z}_i = \begin{cases} \text{Prob}(Y_i = j) = \text{Prob}(X = j) & \text{for } j = 0, ..., L_S - 1 \\ \text{Prob}(Y_i = j) = 1 - \sum_{k \neq L_S} \text{Prob}(Y_i = k) & \text{for } j = L_S \\ \text{Prob}(Y_i = j) = \text{Prob}(X = j) \frac{L_S T + \sum_{k=1}^{k=j} t_k}{(L_S + j)T} T & \text{for } j = L_S + 1, ..., L_S + A \end{cases} \tag{7}$$

The scalar adjustment used for the new arrivals accounts for the decreasing amount of time available because the arrivals occur during time interval. The adjustment factor must be less than one since $t_j < T$. Again, the probabilities are adjusted accordingly to ensure that the probabilities sum to one.

2.5.4 Service Estimation - Approach #3: Scaled Service Rates

The third service estimation approach is a blend of the first two approaches while primarily focusing on scaling the service rate when calculating the associated service probabilities. For the entities arriving within an interval, this approach scales the service rates based on expected time of arrival. Based on $\mu = rt$ with r expressed in units of $\frac{1}{T}$ at time t, the service rates are a product of the time interval size. Therefore, a shortened service rate accompanies a shortened time interval in the same proportion.

Again, the X random variable is used for entities in the system at the beginning of the time interval (L_S).

The random variable $X_j^{''}$ is defined for entities arriving during time interval j with adjusted service rate based on expected average time available. This random variable follows a Poisson distribution such that:

$$X_j^{''} \sim Poisson(\mu \frac{L_S T + \sum_{k=1}^{k=j} t_k}{(L_S + j)T} T) \tag{8}$$

Service Estimation Approach #3: The probability density function for service completions is:

$$Z_i = \begin{cases} \text{Prob}(Y_i = j) = \text{Prob}(X = j) & \text{for } j = 0, ..., L_S - 1 \\ \text{Prob}(Y_i = j) = 1 - \sum_{k \neq L_S} \text{Prob}(Y_i = k) & \text{for } j = L_S \\ \text{Prob}(Y_i = j) = \text{Prob}(X_i^{''} = j) & \text{for } j = L_S + 1, ..., L_S + A \end{cases} \tag{9}$$

2.5.5 Service Estimation - Approach #4: Markov Transitions

This approach for estimating the service probabilities models state transitions within a time interval as a Markov Chain. A Markov Chain is a Markovian process whose state space is discrete, while its time domain T may be either continuous or discrete [5]. This discrete Markov process is memoryless, meaning that the progression

though the states within the Markov chain are independent from one another and thus determined solely by the present state of the process. This discrete process may then be expressed as $X(t), t \in T$ and $P[X(t_{n+1}) = j|X(t_1) = i_1, ..., X(t_n) = i_n] = P[X(t_{n+1}) = j|X(t_n) = i_n]$, where $t_1 < ... < t_n < t_{n+1}$ for any $n > 0$ and state j within the state space. Simply, at each stage, the probabilities for each possible state in the system is the sum of probabilities of transitioning to that present state times the probability of being in the former state.

For the M/M/1 queue example, this Markov process progresses through each time interval using arcs to represent the service quantity (transition to new state) and nodes as the probability of being in each state, which is the number in the system. The summation of all possible combinations arriving at a particular state node represents the probability for a new state transition. The end result of this process produces the state probabilities at the end of each time interval.

Since arrivals follow an exponential distribution, the expected spacing of the known number of arrivals within a time interval are uniformly spaced in $\frac{T}{A+1}$ increments. These time increments are the standard transition points for the Markov chain for all L_S and A combinations. Figure 2 displays an example of an M/M/1 queue time interval where the length of the system is one and two arrivals were generated ($L_S = 1, A = 2$). The first arrival is assumed at the expected time $\frac{1}{3}T$, and the second arrival is assumed at expected time $\frac{2}{3}T$. The end result of this transition process results in the pdf end state probabilities for states 0, 1, 2, and 3. The pdf probabilities are then used to generate a cdf in which the pseudo-random draw is compared to generate a new L_S system length value for the subsequent time interval. The issue arises with how to properly handle the residual probability remaining from the truncated probability tables for each state transition.

Random variable Y_i is defined as the probability of transitioning out of system

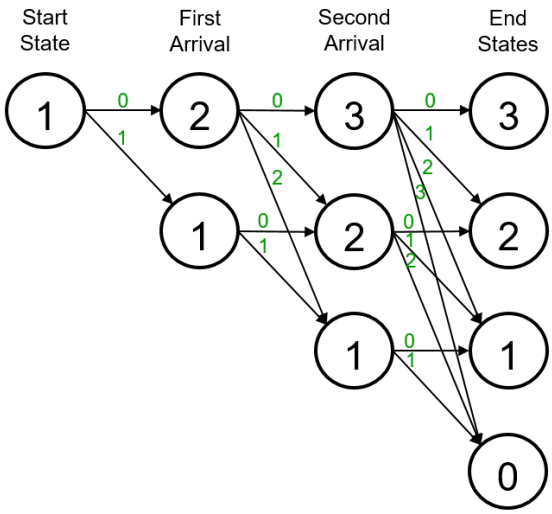

Figure 2. Markov Chain Example: M/M/1 queue with $L_S = 1, A = 2$

state i (i.e. service probability). Random variable X''' follows a Poisson distribution that is adjusted to account for the Markov transition segments within a time interval such that:

$$X''' \sim Poisson(\tfrac{\mu}{A+1}) \tag{10}$$

A truncated service probability table is created in each step transition from each possible states along with its service transitions. Each service probability table has residual probability calculations from the service states that are not allowed. The system may only service up to $L_S + A$ entities within a full time interval, resulting in residual probabilities from states $(L_S + A + 1)$ to ∞. Because pdf tables must sum to one, this residual value needs to be redistributed back into the pdf probability table containing only allowed service states.

This research explores two redistribution techniques for the residual probabilities for states that are not viable. The first technique includes the extra probability into the highest service state assuming that if the system could serve more that it would serve the most possible. The second technique redistributes the residual

probability as a proportional spread of the pre-calculated state probabilities. The service state calculations are then normalized to determine what proportion of the residual probability that it will absorb.

Residual probability redistribution #1 (highest state):

$$Y_i = \begin{cases} \text{Prob}(Y_i = j) = \text{Prob}(X''' = j) & \text{for } j = 0, ..., i-1 \\ \text{Prob}(Y_i = j) = 1 - \sum_{k=0}^{k=i-1} \text{Prob}(X''' = k) & \text{for } j = i \end{cases} \tag{11}$$

Residual probability redistribution #2 (proportion spread):

$$Y_i = \left\{ \text{Prob}(Y_i = j) = \frac{\text{Prob}(X''' = j)}{\sum_{k=0}^{k=i} \text{Prob}(X''' = k)} \text{ for } j = 0, ..., i \right. \tag{12}$$

The predetermined system length (L_S) and number of arrivals (A) affect the system's progression and the Markov transition structure. Random variable Z_j represents the number of entities serviced within a time sub-interval of $\frac{T}{1+A}$. Z_j is conditioned on the predetermined L_S and A values at the beginning of the interval. This approach focuses on the probability of transitioning from a state, so the notation for service probabilities changes to Z_i, where the states transition from state i to state j. Thus, the service probability from transitioning out of a state i and into state j for each of the Markov stages takes the form:

$$Z_i = \text{Prob}(Y_i = i - j) \tag{13}$$

where Y_i is sampled from one of the two residual redistribution techniques. These calculations are repeated for each of $A + 1$ stages of the Markov process.

S_j represents the amount of entities remaining at the end of a time interval. The new system length for the next time interval is equal to the result of the current state, so $L_S = S_j$. Segments, denoted a, represents a predetermined portion of

the Markov transition process prior to state calculations. The amount of transition segments within an interval divides into $A + 1$ segments so $a = \{0, ..., A + 1\}$. The example shown in Figure 2 has $A = 2$, so there are $A + 1 = 3$ segments within the process. Calculating the probabilities of being in each state at the end of time segment is important for each step in this process. The Markov process updates state probabilities (nodes) one segment at a time while considering the prior transitioned states. This process begins with initializing the beginning state node L_S with a probability of 1 because it is a known set value for each time interval.

Service Estimation Approach #4:

Initialize $\text{Prob}(S_0 = L_S) = 1$

$$\mathrm{S}_j = \begin{cases} \text{Prob}(S_a = j) = \sum_{i=1}^{L_S+a} \text{Prob}(S_{a-1} = i)\text{Prob}(Z_i = i - j + 1) & \text{for j = 1,...,L}_S + 1 \\ & \text{and } a = 1, ..., A - 1 \\ \\ \text{Prob}(S_A = j) = \sum_{i=1}^{L_S+a} \text{Prob}(S_{a-1} = i)\text{Prob}(Z_i = i - j) & \text{for j = 0,...,L}_S + A \\ & \text{and } a = A \end{cases} \tag{14}$$

To obtain a probability distribution of end states based on TOS results, the probability of system length L_S is estimated by a frequency count at the end of the time intervals.

2.6 Dynamic Bayesian Network (DBN)

The third modeling technique constructs a Dynamic Bayesian Network (DBN) to serve as a comparison to the other simulation techniques. A Bayesian Network graphically presents the joint distribution of several random variables. A DBN takes the form of a direct acyclical graph (DAG) containing a probability distribution table for each node in the system [13]. That is, all of the edges in the graph contain direction without the possibility of cycling.

Within the DBN process, calculations update node probability tables using Bayesian statistics in which prior nodes influence state probabilities through conditioning. Each node represents a random variable while the arcs connecting nodes indicate the relationship and thus influence between variables. For each step calculation, the prior (parent) nodes in the network influence the (child) nodes in which they connect. Similar to influence diagrams, this network structure maps out relational dependencies for system entities.

Each node in the network represents a user-defined system state. This definition of a state node depends on the unique system or characteristic of interest.

A conditional probability table (CPT) expresses each node for a finite set of mutually exclusive states. The CPT contains probability values for every possible state and takes the general form:

$$\text{Prob}(X_j|\Pi_j) \tag{15}$$

where X_j is a random variable for state j and Π_j represents all parent nodes to X_j.

Nodes without parents contain their prior probability distributions [17]. A simple DBN is displayed in Figure 3, which includes the parent random variable A_1, along with its children B_1, B_2, and B_3. To numerically present the network's joint distribution, the CPT contained in each node must sum to one and contain an exhaustive list of possible state values.

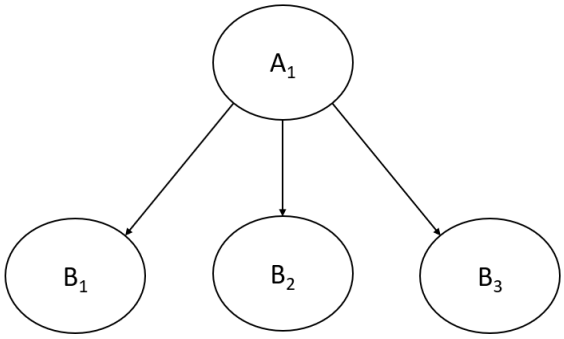

Figure 3. Dynamic Bayesian Network. (probabilities are omitted)

Subject matter experts (SME) or collected data typically help form the network's development and structure [12]. Eliciting all required information from SMEs may often prove difficult, especially for system with large networks and dependencies. SME probability elicitation can be subjective and susceptible to cognitive or motivational bias [9]. Keeney states that a common issue with elicitation from experts concerns the feasibility of the elicitation process. For the elicitation process to produce meaningful output, SMEs should be properly trained in the elicitation process. For effective results, the this elicitation process should include: the problem scope, process of SME selection, multiple SME inputs, proper SME training, lengthy elicitation interviews,

and extensive documentation [7]. Involving so many factors, SME input may be infeasible due to the large number of technical estimates that vary among numerous SMEs in a complex problem scope.

Due to possible issues regarding SME elicitation, probability estimations from collected data points are preferred when available and feasible. To help properly observe the true underlying probability distribution, frequency calculations serve as a means to achieve the estimates. The networks serve as a way to analyze the joint distribution of variables with their respective dependencies through computationally effective algorithms.

In the network, if there exists an edge from X_i to X_j, X_i is defined as the parent of X_j. The set of all parents of X_j is defined as its parent set, denoted by Π_j. $\text{Prob}(X_i|\Pi_i)$ represents the conditional probability distribution for the network N containing the attribute set χ and is defined as follows [10]:

$$\text{Prob}_N[\chi] = \Pi_{i=1}^{n}\text{Prob}(X_i|\Pi_i) \tag{16}$$

In the simple example from Figure 2, the defining probability is:
$\text{Prob}_N[\chi] = \Pi_{i=1}^{3}\text{Prob}(B_i|A_1) = \text{Prob}(B_1|A_1)\text{Prob}(B_2|A_1)\text{Prob}(B_3|A_1)$.

A major benefit of Bayesian Networks comes from the ability to inference. Inference is the task of computing the probability of each value of a node in a network when other variables' values are known [18]. After the completion of all node conditional probability calculations, an analyst is able to extract different "what-if" analyses, furthering the network's usefulness. Since all possible permutations of system nodes must be computed within the network, the DBN model is often initially very time consuming and computationally expensive to calculate each unique node dependency along with its associated probability distribution. Once all of the probabilities are estimated however, the DBN is extremely useful and may offset the initial computa-

tional effort since no additional derivations are required for "what-if" analysis. Unlike simulations, Bayesian Networks do not require replications. The network maintains the probability distribution for each node, thus allowing the user to update the system probabilities for desired outcomes or starting states. DBN model formulation must define all possible system states and node conditioning within the network. Due to many branches within the network and assumed slow computational run-time, the DBN may not be considered as a viable modeling technique for complex systems. For instance, if there exists an arbitrary system that begins with three states and every node in the system may branch out into three additional states, the amount of node computations grows at the rate of 3^n where n is the amount of steps in the system. Even though this is initially computationally expensive, it is possible that it is still beneficial in the future where multiple "what-if" scenarios are needed. This also serves in a case requiring reverse inferencing where parent node state probabilities serve as the subject of interest.

A common method for assessing the underlying distribution of a DBN is through frequency tables. One of the challenges faced by analysts is how to properly split continuous variables into discrete bins [11]. The selection for the number of bins can greatly affect the theorized distribution from the model output. A common method for choosing these bins is assigning an equal number of cases to each bin or assigning equal-sized bins. Equal-sized bins may result in CPTs that contain uninformed probabilities due to missing or scarcity of data, which adds computational run-time to states that do not affect the model performance or overall outcome. Even if a discretized bin has insignificant probability of occurrence, the computations are still conducted with this method of bin definition. The higher number of bins may help identify the proper distribution but comes at the trade off of computation time. When determining state probabilities, too many bins may lead to poor model quality

due to insufficient data contained within each bin of the probability distribution. Too few bins however, may also limit the model's ability to represent the probability distribution as a result of low resolution.

This is not specifically applicable to the discrete M/M/1 queue, but useful for networks containing continuous probability distributions.

The development of the DBN for the M/M/1 queue example is similar to the TOS modeling technique. The DBN uses user-defined time intervals however, rather than generating a pseudo-random number to determine simulated outcomes, the DBN incorporates all probabilities as the likelihood of taking a particular arc in the network. If the time interval duration is increased in size, then more events (arrivals or services) occur within the interval for the same rate. For example, if $\lambda = 1$ for time interval size of $T = 1$, then $\lambda = 1.5$ has the same rate for time interval size of $T = 1.5$. The probability of transitioning from a state (number in the system, L_S) to another state is the sum of the probability paths between those states. For example, the probability to transition from state 1 ($L_S = 1$) to state 2 ($L_S = 2$) is the sum of probabilities of one arrival with no service completions and two arrivals with one service completion, and so on. This infinite stream of probabilities may only be calculated for those feature combinations that have a significant probability within a specified tolerance. Instead of replications, the DBN calculates and maintains all state probabilities while progressing through the model. The DBN process has achieved steady state when the probabilities stabilize between time intervals. The initial starting condition may affect the time required for these probabilities to stabilize. Once probabilities converge to steady-state, the end product takes the form of a probability table for the likelihood of being in each end state.

2.6.1 DBN Process - Service Estimation Approaches #1 through #3

The service probability estimation approaches are identical to that of the time-oriented simulation. The most accurate estimation approach for the conditional service probabilities will be chosen for DBN model execution, specifically for the generation of the unique CPTs. As described earlier, the DBN model does not require replications because it stores all probabilities while continually updating CPTs as the model progresses. The DBN is an iterative process where it constructs a probability table for all possible system states.

The process for calculating state probabilities are broken down into four primary steps: initial, arrivals, services, and end. The inital CPT contains probabilities of being in each system state (L_S) at the beginning of the time step iteration. For example, if the system is beginning at time t_0 while empty and idle, the CPT will have a 1.0 probability of being in state 0 while all other states have probability 0. The arrival CPT follows a discrete Poisson distribution where probabilities remain constant for each iteration due to a fixed arrival rate λ that is independent of external entities or processes. The service CPT however contains variability because this probability table is dependent on the system state probabilities for both the initial as well as the arrival CPTs. Since the initial CPT varies per iteration, it causes the service CPT to vary in possible states. Service probability estimations are most crucial because they directly affect model results. This process may use any of service estimation approaches #1 through #3 when deriving the service CPTs. With the initial, arrival, and service CPTs formed, the DBN process then calculates the end CPT based on the information prior. The end CPT derivation treats all of the inputs (initial, arrivals, and service CPTs) as independent, resulting in a simple multiplication of CPT state combinations. For instance, there are multiple ways to end in state 0 which include (initial state=0, arrivals=0, services=0) or (initial state=0, arrivals=1, services=1).

All possible combinations have associated probabilities and summed together to update its respective end state. The end CPT then serves as the initial CPT for the next iteration in the DBN process. This iterative process continues until system state probabilities converge to steady-state values. A user-defined convergence tolerance level serves as the model termination criteria, which may change due to system specifications. A relatively small convergence tolerance threshold is highly encouraged but may require more computational run-time.

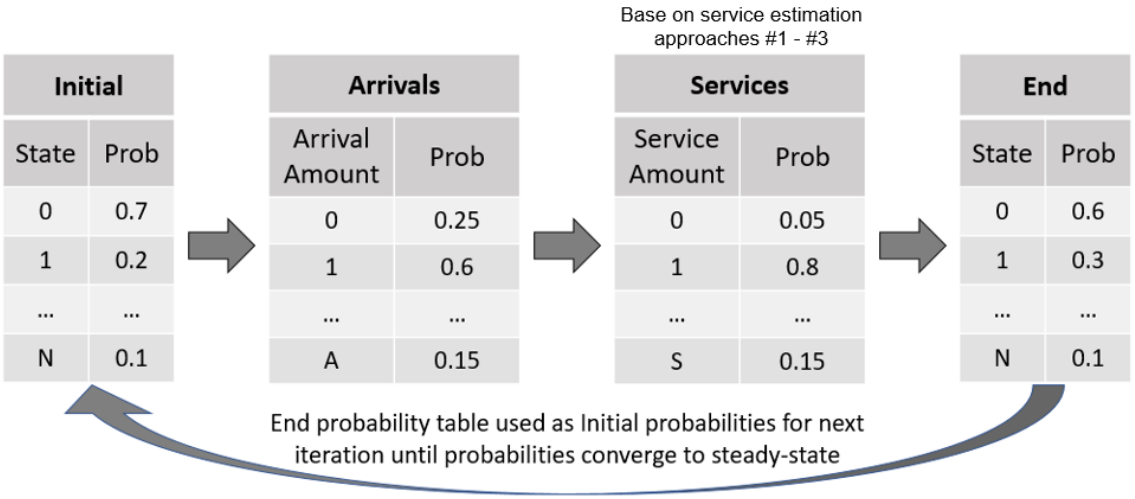

Figure 4. Dynamic Bayesian Network Process Flow: Approaches #1 - #3 (probabilities displayed are not calculated and intended for process demonstration only)

2.6.2 DBN Process - Service Estimation Approach #4 (Markov Transitions)

Implementation of the service estimation approach #4 within the DBN process is nearly identical to approaches #1 through #3 but embeds the service distributions into the Markov transitions. The primary difference of this approach emerges with the structure of the Markov transition process. Each Markov transition accounts for the service rates, combining the service CPT and end CPT together within the transition steps of a time interval. Figure 5 highlights these details for the DBN

process as the service and end state CPTs exist simultaneously within the Markov transition process.

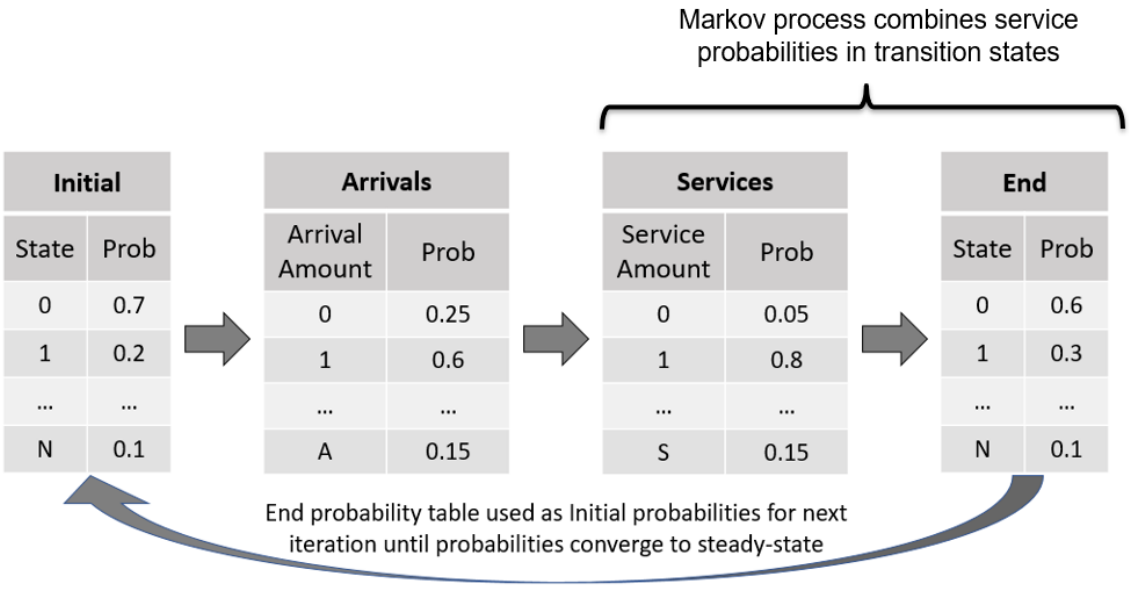

Figure 5. Dynamic Bayesian Network Process Flow: Approach 4 (probabilities displayed are not calculated and intended for process demonstration only)

To obtain a probability distribution of end states, the probability of system length L_S is simply derived directly from the final end probability distribution due to the unique processing of the DBN model.

2.7 Model Comparisons

The system example using an M/M/1 queue is beneficial to this research because this discrete system is well researched, containing theoretical steady-state values which serve as a baseline to calculate overall model accuracy. Comparing model output to the system's theoretical state value produces an error value which then serves as a quantifiable metric to assess the model's accuracy and precision. The system characteristic of interest is the system length (L_S) and thus defines states in the model. The outputs include mean and variance calculations, where theoreti-

cal steady-state mean is represented as $\frac{\rho}{1-\rho}$ and theoretical steady-state variance as $\frac{\rho}{(1-\rho)^2}$.

From queueing theory, the length of the system, L_S, depicted at steady-state $L_S = \frac{\rho}{1-\rho}$, where $\rho = \frac{\lambda}{\mu}$, serves as the primary system metric of interest. Each modeling technique uses the system state definition as discrete counts of L_S. A probability table of possible state values may be produced from each technique's results. Table 2 shows the different approach for calculating the average end state probabilities.

Table 2. Probability of M/M/1 System Length Estimation

DES	Percent of simulated time
TOS	Relative frequency
DBN	Final probability upon convergence

The sum product of the state values times their associated probabilities produces an estimate for the steady-state length in the system L_S. Each modeling technique contains unique characteristics that may affect the model's accuracy. Table 3 illustrates possible factors that may affect the model's development and performance.

First, the beginning state of the system is important to consider because as λ and μ rates increase, the model run time may also need to increase to ensure that the modeled system reaches steady-state behavior. In this research, each model will begin empty and idle without a warm up period, meaning that the modeled system will begin on time t_0 with zero entities in both the queue and server. The amount of replications is also considered for both simulation techniques because they utilize pseudo-random numbers, causing each replication to contain a unique set of outputs. The DBN model however, does not require replications as it is a single pass through the model. The DBN terminates execution when the state probabilities between time intervals converge within a specified convergence tolerance level.

All of the models have to handle probability table truncation due to disallowed

combinations between the system state and amount of possible services. Each service estimation approach addresses this probability distribution truncation. Finally, the time interval size affects both TOS and DBN models. As the time interval size duration increases, the likelihood of multiple events occurring during that time also increases. This has the possibility to cause increased variation in the results due to a higher amount of events occurring simultaneously.

Table 3. M/M/1 Queue Characteristics

Conditions	**DES**	**TOS**	**DBN**
Beginning State	✓	✓	✓
Replications	✓	✓	n/a
Convergence Tolerance	n/a	n/a	✓
Truncation Amount	✓	✓	✓
Time Interval Size	n/a	✓	✓

Each model results in a table representing the overall probability of being in a particular state. For DES, the percent of time of being in each state determines the probability of states. For the TOS, the relative frequencies of system states within the duration of the model estimates the overall end state probability distribution. Since the simulation techniques require replications, the final step calculates state frequencies from a matrix containing all stored values for state counts across replications. Conducting the frequency calculations in the final step helps reduce rounding error that may be present when compared to taking the average of averages. The end state probability distribution output from the DBN model is extracted directly from the end CPT upon model convergence.

III. Results and Analysis

3.1 Model Development

Python 3.6.5 serves as the primary coding platform for each modeling technique. Model execution uses a Windows 10 computer with 16GB of physical memory and an Intel i7-8550U CPU @ 1.80GHz. The DES model uses SimPy 3.0.11, which is a free-to-use python module published under The MIT Licence. SimPy is a simple discrete-event simulation framework that utilizes python's base computing environment. Both TOS and DBN models also execute python code, developing from the ground up for this research. Aggregating all models into one master python file enhances usability and also consolidates each model's output between runs.

Since the DES model did not require a service probability estimation, this model did not require different versions of development. The TOS model however, requires many versions to test and analyze each service estimation approach along with calculating the lower and upper bounds. TOS model development was an iterative process where each service estimation approach was an evolution from the one prior. The theoretical mean was used to determine the accuracy of each approach. Verification includes analyzing the amount of simulated arrivals and services within each time interval while ensuring that the overall expected behavior is proper. For instance, if $\lambda = 0.1$, then 10 total arrivals are expected in 100 time intervals. The service estimation approaches use the TOS modeling technique for development. The DBN then adopts the most appropriate technique based on accuracy. Since both TOS and DBN models utilize the same service estimation approach, both are expected to produce similar accuracy.

3.2 Service Estimations

With the exception of service estimation approach #4, Markov transitions, each approach is an evolution of the one prior. Initial test model runs considering the first three approaches appears to estimate the services generations well. The results of additional test cases with varying λ and μ values indicate that the overall TOS models were estimating service states poorly, in particular cases. Cases where the inter-arrival are very small, the estimation approaches are seldom used. As the arrival rates increases, the service completion approximations are used more frequently. The results indicate that the estimation approaches did not capture the true characteristic of the M/M/1 queue system. Table 4 shows two sample cases and each approach's estimate of the steady-state theoretical mean of L_S. Out of these three estimation approaches, approach #3 appears to be the most consistent in estimating the steady-state mean. The second case with $\lambda = 4$ has more arrivals per interval,

Table 4. Service Estimation - Mean Comparison (Two Sample Cases)

	Case $\lambda = 0.1$ and $\mu = 0.4$			Case $\lambda = 1$ and $\mu = 4$		
TOS	**App #1**	**App #2**	**App #3**	**App #1**	**App #2**	**App #3**
Calculated Mean	0.353	0.349	0.343	1.060	0.908	0.746
Theoretical Mean	0.333	0.333	0.333	0.333	0.333	0.333

* 2000 time intervals, time interval size 1

and hence, used the service completion approximations more frequently. All three of these approaches served too few of the arrivals within an interval, which results in the calculated mean of the number in the system L_S to be too high.

Understanding that there still exists error in the first three service estimations, the development of a fourth approach begins which involves treating each time interval as a Markov transition process. This Markov process takes in consideration all possible

service states as well as feasible end states, which are all conditioned on the status of the system (L_S and A) for that time interval.

Considering calculations of service probabilities, there exists an issue with truncated probability distributions due to disallowed service amounts. The probabilities for possible service amounts must sum to one within each Markov transition. After service probability calculations for allowed service states, there still exists a residual probability amount which requires redistribution among the possible service states. The first theory includes the residual probability into the highest service state. For instance, if the service probability table contains service states 0, 1, and 2, then service state 2 receives the remaining residual probability amount, which ensure that the probabilities sum to one. The second theory redistributes the residual probability proportionally into the state values. Table 5 displays two sample outputs with service estimation approach #4 for each residual probability redistribution technique. While not perfect, each of the techniques performed fairly well when considering model output compared to theoretical steady-state values. The first technique, where the last state receives the remaining probability, is less accurate as the service rate parameter increases.

Table 5. Residual Probability Comparison (Sample Cases using Markov transitions)

	Case $\lambda = 0.1$ and $\mu = 0.4$		Case $\lambda = 1$ and $\mu = 4$	
TOS	**Include in Last State**	**Proportional Spread**	**Include in Last State**	**Proportional Spread**
Calculated Mean	0.338	0.367	0.240	0.309
Theoretical Mean	0.333	0.333	0.333	0.333

* 2000 time intervals, time interval size 1

3.3 Model Accuracy

For each unique combination of λ and μ, accuracy measures the error value when comparing the output mean and variance to their respective theoretical steady-state values. Here, $\frac{\rho}{1-\rho}$ represents the theoretical steady-state mean and $\frac{\rho}{(1-\rho)^2}$ as theoretical variance, where $\rho = \frac{\lambda}{\mu}$. To analyze each modeling technique under different conditions, parameters are varied to help identify true model performance. These parameters include λ, μ, replications, number of entity generations (arrivals), amount of time intervals, time interval duration size, and convergence tolerances. Table 6 indicates which modeling technique is influenced by each parameter. All models execute with adequate duration which ensures that each model properly obtains steady-state characteristics.

Table 6. Varied Parameters by Modeling Technique

	DES	**TOS**	**DBN**
λ / μ	✓	✓	✓
Replications	✓	✓	n/a
Number of Entities (Arrivals)	✓	n/a	n/a
Amount of Time Intervals	n/a	✓	n/a
Time Interval Size	n/a	✓	✓
Convergence Tolerance	n/a	n/a	✓

Figure 6 displays 32 unique cases where the bars represent a modeling technique compared to the theoretical mean values. The x-axis in this graph displays each unique λ and μ pairing with increasing magnitudes of the parameters. At first glance, it appears that each technique is modeling the system adequately with each bar closely following the theoretical mean trend line. While it appears small, closer inspection shows that there exists a small gap between the calculated bars versus the theoretical line in some cases.

The results for the same 32 test cases are presented as the percent difference of the theoretical steady-state number in system L_S minus the calculated mean value.

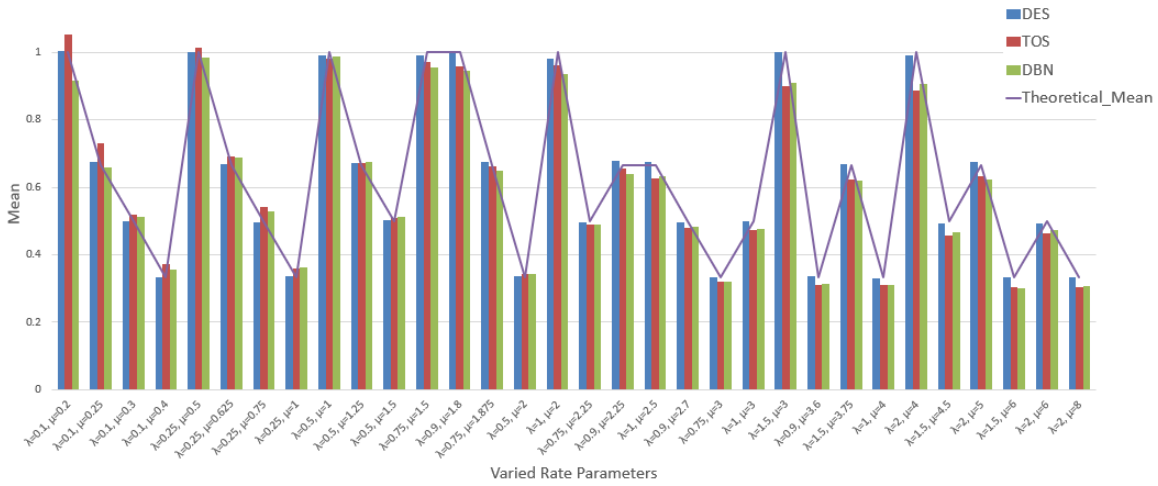

Figure 6. Model Calculated Mean Versus Theoretical Mean

This view scales the error values between each model while adjusting accordingly to the magnitude of the outputs. For example, there is a distinct difference in the error between 0.4 and 0.5 compared to 4.9 and 5.0. While both of these contain a raw difference of 0.1, the proportion of the error is an important factor when assessing model accuracy. Figure 7 displays the same test cases with the model error as a percent difference from the theoretical mean. In Figure 7, a negative value indicates that the calculated mean is lower than the theoretical mean, i.e. indicating that the model generates too many services and thus resulting in a mean value that is too low. The modeling error, whether high or low, is a function of the service completion approximations for entities arriving during an interval. Those approximations on the number of arrivals completing service during an interval occurs more often with a higher rate of service. However, the approximations are applied more often with a higher arrival rate. The services specifically drive model output because this is the portion within the TOS and DBN modeling techniques that uses the estimation approaches, which depends on other internal factors. The arrival probability distribution does not change with model progression because it is independent of modeling factors and states. The inter-arrival rate λ dictates the arrival probability derivations.

For the data shown in Figure 7, a linear regression line can also be properly fit to the data that is significant at the 95% confidence level. This best fit line utilizes service rate μ parameter as the regression's independent variable, and the proportion difference error value as the dependent variable. The resulting best fit line is: $error = 0.0188 - 0.0141 * \mu$, which indicates that as the service rate increases the model error becomes more negative. The increasing error in the negative direction expects to follow the linear trend of the fitted regression line. A fairly easy way to combat this error is to decrease the time interval size because it directly scales the model parameters.

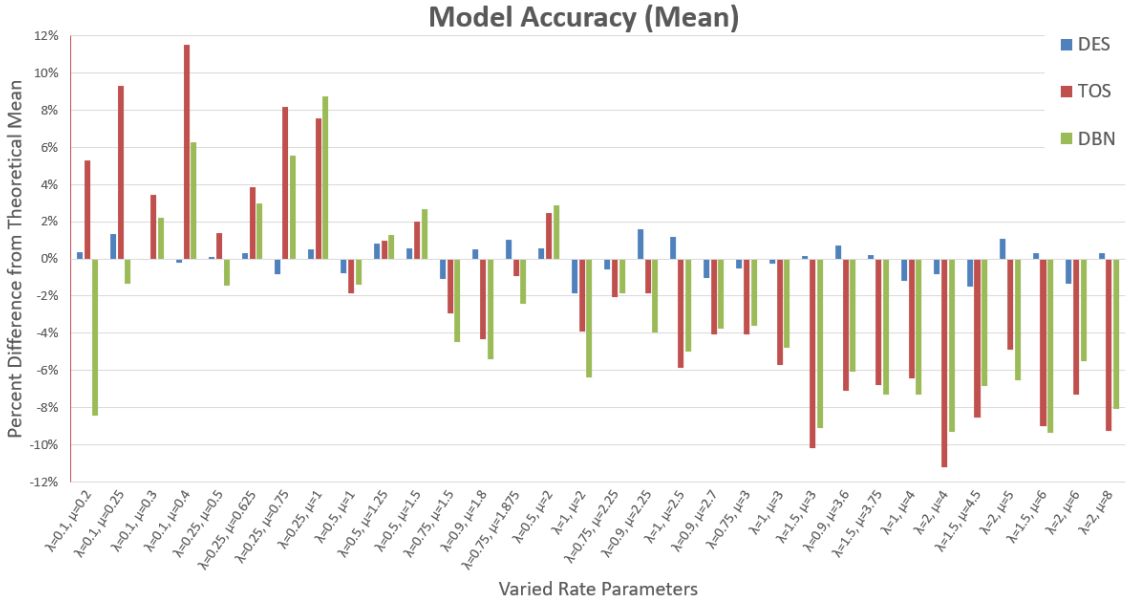

Figure 7. Model Error - Percent Difference from Theoretical Mean

Further exploring model errors includes analyzing the calculated end state probabilities for each modeling technique. Generally, the DES models contain small error values since it represents the system fairly accurately. The system's true end state probability distribution should closely mirror the DES model output. Since there does not exist a true theoretical distribution of the M/M/1 states for comparison, the DES state probabilities serves as a baseline when analyzing the other modeling

techniques. In a specific case using $\lambda = 2$ and $\mu = 8$ shown in Figure 8, the TOS and DBN state 0 probabilities appear to be slightly high while the state 1 probabilities are low. These state probabilities cause the overall mean value to be lower than the theoretical mean, again assuming that the DES output is closest to the desired distribution.

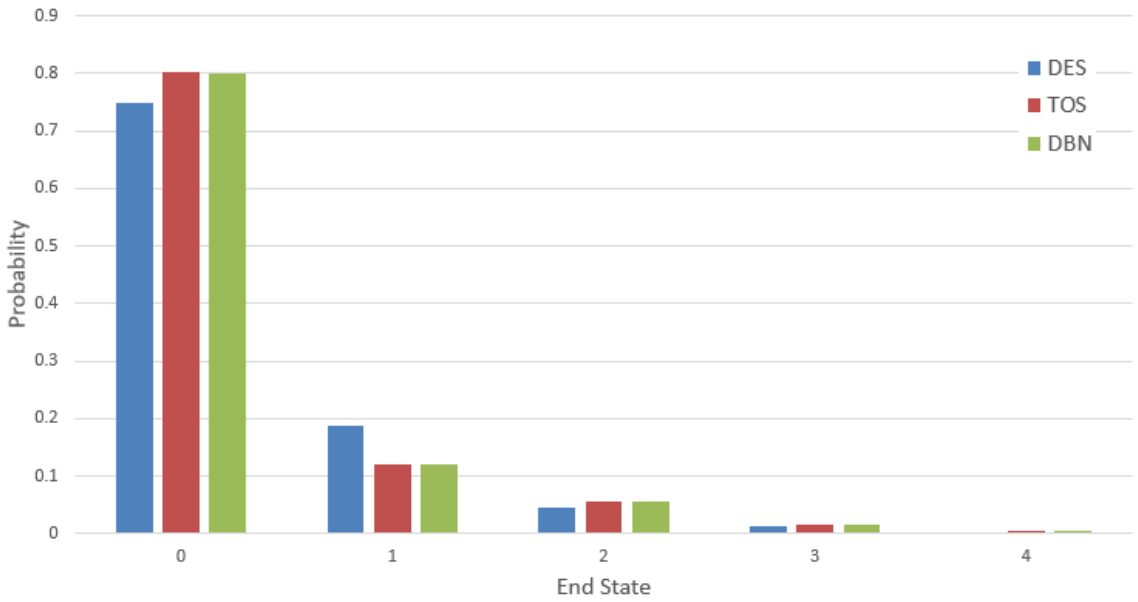

Figure 8. State Probability Distribution ($\lambda = 2$, $\mu = 8$)

To ensure that results are both tractable and comparable, each model executes under proper conditions. That is, each modeling technique requires proper conditions to allow the system to reach steady-state probabilities. DES requires an adequate amount of generated model entities, TOS requires an adequate amount of time intervals, and DBN requires a proper convergence tolerance. Each of these factors will individually affect the models abilities to produce accurate results. This highlights the issue of initialization bias because each model begins with an empty and idle system. Since the M/M/1 queue does not have a known steady-state probability distribution, the DES, as the most accurate approach, serves as the baseline for comparison. Again, the DES model best estimates the steady-state mean, so it serves as

the baseline.

Since the TOS model's termination criteria is the amount of specified time intervals to simulate, this factor may possibly contribute to model error. A model run must be long enough to adequately reach steady-state characteristics. Figure 9 displays the TOS model with Markov transitions to estimate service probabilities for the system containing $\lambda = 2$ and $\mu = 8$. Consistent results with varied time interval amounts indicate that initialization bias does not contribute to the model's error. The state probabilities are unchanged among the differing run lengths, illustrating that initialization bias from an inadequate amount of time intervals does not attribute to model error.

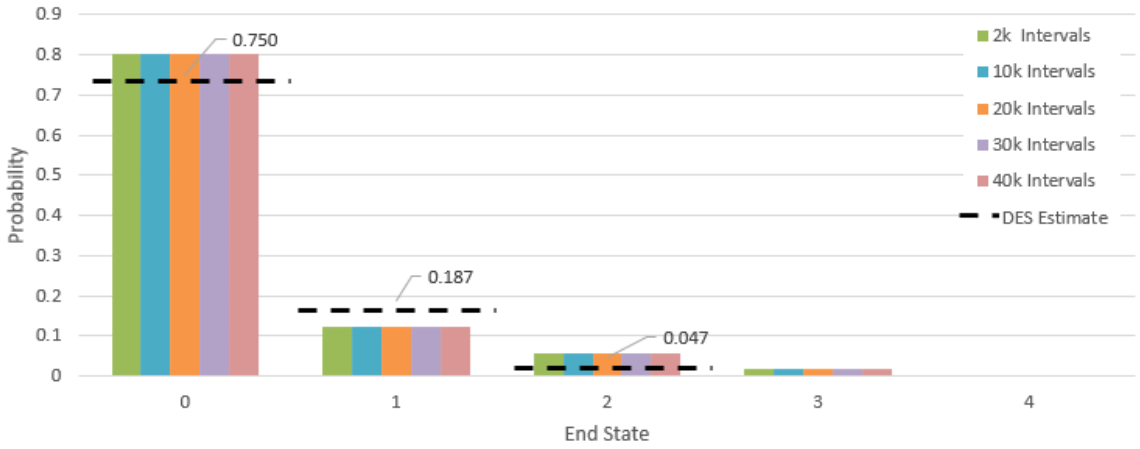

Figure 9. TOS Initialization Bias ($\lambda = 2$, $\mu = 8$)

An initialization bias analysis may also affect the DBN model output. The varied parameter of interest for the DBN model is convergence tolerance, which affects the point at which the model terminates. The DBN output in Figure 10 also indicates that the model parameters do not attribute to model error. Since both TOS and DBN models have nearly identical output, the analysis concludes that the model errors most likely result from service estimations because both models share the same approximation of the embedded Markov Process.

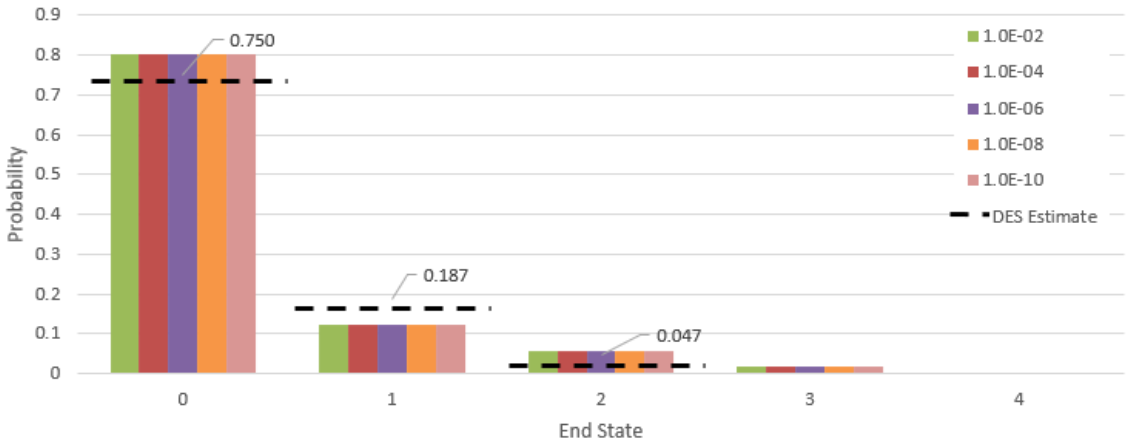

Figure 10. DBN Initialization Bias ($\lambda = 2$, $\mu = 8$)

3.4 Model Run-Time

An additional measure to compare the modeling techniques is to assess computational run-time. This requires a basic assumption for each model because they are each affected by separate termination criteria. To terminate a model, the total number of specified entity generations affects DES while the amount of simulated time intervals affects TOS. The DBN model's run length is dependent on a user-defined tolerance level for end state probability convergence. Due to difficulty in properly scaling each of these factors to a similar magnitude, each model stopping criteria is set to a constant number of entities so that the rune-time values are adequate for all cases.

The DES model consistently has the quickest run-time among the modeling techniques. Each instance of this model generates 5000 entities at 30 replications and is unaffected by the varied λ and μ parameters. For DES, the model parameters do not affect run-time generation of 5000 entities. Model entity arrival and service times each follow an exponential distribution with their associated rate parameters. The higher parameter values do not affect the DES model's total amount of required computations, but do increase the simulated time.

Figure 11. Model Run-Time (DES: 5000 entities, TOS: 2000 time intervals, DBN: 0.00001 tolerance)

The TOS model was among the slowest techniques while executing a simulation consisting of 2000 time intervals with a time interval length of one and 30 replications. This model's run-time varies around 0.8-1.1 minutes for each unique test case. A possible explanation for this technique's relatively poor run-time results may be due to the model's construction. The model requires a loop through each of the 2000 time intervals while evaluating probabilities and pseudo-random draws within each interval. In an attempt to increase code efficiency, each unique probability distribution calculates once and then stored in a matrix for future reference. Figure 11 indicates that there exists a gradual upward trend in run-time as the system parameters increase. The analysis expects this behavior because as the λ and μ parameters increase, the higher the chance of multiple occurrences within the same time interval. Increased rate parameters creates more variability, but also higher amount of probability calculations due to an increasing length of possible state values.

The DBN model's stopping criteria is a fixed probability convergence tolerance of 0.00001. This value is set small enough to ensure that the probabilities reached a steady-state value without the chance of coincidence. For small system parameters, this model performs similar to that of DES. As the system parameters increase, the

run-time also increases due to longer probability tables. This technique calculates separate probability distributions for the start state, arrivals, services, and end states. To determine the end state distribution within each time interval, the model evaluates the probability tables and retains all possible combinations between states. Large λ and μ values result in long probability tables for all possible states, requiring more computations in each step iteration for the Bayesian network process. This higher amount of possible state values implies that the amount of state computations also increases.

There exist scenarios when time-oriented simulations are preferred over discrete-event simulations. Specifically, combat and agent-based modeling techniques use a time-series progression. A specific example of such a situation is if a manufacturing plant is trying to assess a system's state at any given time. Even though rates and output numbers are known, the entire system requires analysis to ensure that each section is running optimally. A discrete-event approach may produce the overall behavior of the system while a time-oriented approach may provide additional behavioral detail to a specified level of aggregation. Under certain conditions, a DBN model may serve as an adequate alternative to the TOS approach. The DBN model also has the strengths of reverse inferencing for further system exploration because all of the possible state probabilities have been calculated and stored after model execution.

IV. Conclusion

4.1 Summary

This research explores the development and execution of three separate modeling techniques. The three models of interest include a discrete-event simulation, time-oriented simulation, and a dynamic Bayesian network. Each modeling technique has its own definition and method of execution to estimate system outcomes.

The DES model is an object-oriented approach where the generation of system objects trigger model progression. This implies that the DES model has unique random outcomes and require enough system entities as well as replications to properly estimate steady-state values.

The TOS model is a time series approach with fixed intervals in which probabilities of system states are estimated within each time interval of the model progression. A pseudo-random draw compares to a state probability distribution, which in turn determines the system state within each interval. This process in the TOS model continues for a user-specified amount of time intervals. Similar to DES, the TOS model also contains unique outcomes and requires an adequate amount of time intervals and replications to properly estimate system steady-state behavior.

The last model is the DBN, which also utilizes fixed-width time intervals similar to TOS. The DBN model however, does not utilize a random draw to determine state values but rather maintains all state probabilities and updates the distribution based on the time interval prior. The DBN process executes until the state probabilities converge within a user-defined tolerance level and thus does not require replication.

To compare modeling techniques, an M/M/1 Queue example is chosen because it is a discrete system that has closed-form theoretical steady-state values for system behavior. Each model defines system state as the amount of entities in the system

(L_S). The steady-state values serve as a baseline for model comparison of accuracy.

During model development, analysis reveals that the TOS and DBN models require service probability estimation for arrivals during time intervals. This service estimation is necessary because though there is a known mean service rate (μ), the amount of services are dependent on the current system state. In any given time interval, the system cannot serve more entities than currently exist. This creates a situation with truncated probability distributions. Since exact time of arrivals are unknown, each arrival generation requires an assumption to arrive in the system at its expected time. This arrival assumption leads to the development of four different service estimation approaches. The fourth approach treats the progression through a time interval as Markovian transitions with an assumed time for each arrival. Among all service estimation approaches, the fourth approach produced the most accurate and consistent results when modeling average system length.

Among the three modeling techniques, the DES model consistently outperforms the others in both accuracy and run-time. The DES has the lowest calculated error when modeling average length of the system and was also the quickest to execute when considering computation time. The TOS and DBN models have similar accuracy results because they were both using the same service estimation technique. As rate parameters or time interval size increase, the TOS and DBN models continue to underestimate the average length of the system. In cases where the calculated average system lengths are lower than their theoretical values, this indicates that the service estimation approaches are generating too many services for the entities arriving within the time interval. Especially when the rate parameters or time interval sizes are small, the DBN model executes quicker than the TOS model. In a correct set of circumstances, if a system requires a time step approach, a DBN model may serve as an appropriate alternative to simulation. The results of this research demonstrate that

a Bayesian Network can produce a distribution of outcomes similar to the distribution from multiple replications of simulations.

4.2 Future Research

This research may serve as a baseline for further development and refining of each of these modeling techniques. The service probability estimations have error in the formulation, especially when rate values increase. This is an area that may be further developed to improve on the estimation approach. These models may also be implemented using different systems, mainly other queueing models containing multiple servers and possibly different arrival and service distributions. A major benefit of Bayesian networks is the possibility for inferencing. Once a DBN model is computed, the state probabilities are stored, thus having the ability to be used in different what-if scenarios without executing a new instance of the model. Further research may benefit from exploring Bayesian inferencing further for additional insights and overall usefulness.

Further development of the DBN model may be devised to represent a system with continuous distributions. An example of such an application may be to develop the DBN for reliability models to assess some component's expected lifetime and mean time to failure. The adaptation to continuous cases gives rise to the need to evaluate distributions. One focus may be to assess the trade-off between using fixed-width bins versus fixed-probability bins when evaluating continuous distributions. In complex systems, DBNs are often initially computationally expensive because they calculate all possible permutations of system states. Using fixed-width bins has the possibility to use computational power on the distribution tails, even for insignificant probability. Implementation of fixed-width bins may help improve DBN execution time while preserving model accuracy.

Appendix A. Python Code

```
'''================================================================='''
'''''''''                    Capt Aaron Salazar                    '''''''''
'''''''''                       M/M/1 Queue                        '''''''''
'''          Master code containing all models - DES, TOS, DBN          '''
'''TOS and DBN service estimation based on approach #4 (Markov Process)'''
'''================================================================='''

''' Residual service probs within markov process is spread proportionally '''

#Library -- C:\Users\Aaron\Desktop\Thesis

#########################################################################
#######     Required Packages and Parameter Definitions      #######
#########################################################################
import random
import simpy #SimPy 3.0
import pandas as pd
import numpy as np
from scipy.stats import poisson
import timeit

model_list =  ["DBN"] # inputs: ["DES","TOS","DBN"]

output_decision = 'No' # 'Yes' or 'No' if we want to output excel file
outfile = "master_output_markov(proportional res)_5.xlsx"

lambda_list = [.1,.25,.5,.75,.9,1,1.5,2]
mu_mult_scalar_list = [2,2.5,3,4]

replications = 30     # DES and TOS

NUM_CUSTOMERS = 5000 # DES ONLY - Total number of customers

time_end = 2000       # TOS ONLY - Total time
step_size = 1         # TOS and DBN

conv_tol = 0.000001  # DBN ONLY

n=0
#########################################################################
#########################################################################

for model in range(len(model_list)): #run the models specified

    if model_list[model] == "DES":
        '''#############################################################################'''
        '''##################  Discrete-Event Simulation (DES)  ##################'''
        '''#############################################################################'''
        mean_list = []
        var_list = []
        my_lambda_list = []
        my_mu_list = []
```

```
53          theoretical_mean_list = []
54          theoretical_var_list = []
55          raw_diff_list = []
56          abs_diff_list = []
57          proportion_diff_list = []
58          overall_sys_size_list = []
59          state_probs_list = []
60          run_time_list = []
61
62          loop_counter = 0
63          for lambda_loop in range(len(lambda_list)):
64              for mu_loop in range(len(mu_mult_scalar_list)):
65
66                  start_timer = timeit.default_timer()
67                  loop_counter += 1 #used to print loop counter while running code
68
69                  for r in range(replications): #amount of replications
70
71                      RANDOM_SEED = random.seed(5,95)
72                      n = 0 #drop first 'n' observations to delay record keeping for each iteration
73                      my_lambda = lambda_list[lambda_loop]
74                      my_mu = my_lambda*mu_mult_scalar_list[mu_loop]
75                      my_rho = my_lambda/my_mu
76                      overall_mean_numinsys = my_rho/(1-my_rho) #theoretical mean
77                      theoretical_var = my_rho/((1-my_rho)*(1-my_rho))
78
79                      #initialize lists to store local values per replication
80                      num_in_sys_list = []
81
82                      queue_time_stamps = []
83
84                      def source(env, number, interval, counter, mu):
85                          '''Source generates customers randomly'''
86                          for i in range(number):
87                              c = customer(env, 'Customer%02d' % i, counter, service_time=random.
    expovariate(mu))
88                              env.process(c)
89                              t = random.expovariate(interval)
90                              yield env.timeout(t)
91
92                      def customer(env, name, counter, service_time):
93                          '''Customer arrives, is served and leave'''
94                          #arrive = env.now
95                          #print('%7.4f %s: Arrived' % (arrive, name))
96
97                          with counter.request() as req:
98                              yield req
99                              #wait = (env.now - arrive)
100                             # We got to the counter
101                             #print('%7.4f %s: Waited %6.3f' % (env.now, name, wait))
102                             svc = service_time
103                             yield env.timeout(svc)
104                             #print('%7.4f %s: Finished' % (env.now, name))
105
```

```
                        num_in_sys_list.append(len(counter.queue)+len(counter.users))
                        queue_time_stamps.append(env.now)

                # Setup and start the simulation
                random.seed(RANDOM_SEED)
                env = simpy.Environment()
                # Start processes and run
                counter = simpy.Resource(env, capacity=1)
                env.process(source(env, NUM_CUSTOMERS, my_lambda, counter, my_mu))
                env.run()

                average_num_in_sys = sum(num_in_sys_list[n:])/len(num_in_sys_list[n:])
                overall_sys_size_list.append(average_num_in_sys)

                '''''''''  Tally exact amount of time in each system-state '''''''''
                sys_info = pd.DataFrame({'Time':queue_time_stamps, 'System_Length':
num_in_sys_list})
                max_size = max(sys_info['System_Length'])

                #initialize prob table up to the max number generated in the system
                time_in_states = []
                for i in range(max_size+1):
                    time_in_states.append(0)

                ### tally times per state
                for i in range(len(sys_info)):
                    if i != 0:
                        temp_state = sys_info.iloc[i,1]
                        time_in_states[temp_state] += sys_info.iloc[i,0] - sys_info.iloc[i
-1,0]
                sys_info_state_times_df = pd.DataFrame({'System_Length Rep%s' % r:
time_in_states})

                ### save counts in each iteration
                if r == 0:
                    overall_sys_state_counts = sys_info_state_times_df
                else:
                    overall_sys_state_counts = pd.concat([overall_sys_state_counts,
sys_info_state_times_df],axis=1)

                ### replace nan's with 0 (nan's happen because of varying state lengths
between reps)
                for row in range(len(overall_sys_state_counts)):
                    for col in range(len(overall_sys_state_counts.columns)):
                        if np.isnan(overall_sys_state_counts.iloc[row,col]) == True:
                            overall_sys_state_counts.iloc[row,col] = 0

                ### convert counts to probabilities
                for col in range(len(overall_sys_state_counts.columns)):
                    prob_temp_list = []
                    col_total = sum(overall_sys_state_counts.iloc[:,col])
                    for row in range(len(overall_sys_state_counts)):
                        prob_temp_list.append(overall_sys_state_counts.iloc[row,col]/col_total
)
```

```
154
155                         end_probs = pd.DataFrame({'State Probs Rep%s' % col:prob_temp_list})
156
157                         if col == 0:
158                             end_probs_df = end_probs
159                         else:
160                             end_probs_df = pd.concat([end_probs_df,end_probs],axis=1)
161
162                     ### calculate overall probs by averaging each row across the columns
163                     final_prob_list = []
164                     for row in range(len(overall_sys_state_counts)):
165                         row_total_temp = sum(overall_sys_state_counts.iloc[row,:])
166                         temp_prob = row_total_temp/sum(overall_sys_state_counts.sum()) #prob is
    rowtotal/totalofentiredataframe
167                         final_prob_list.append(temp_prob)
168
169                     #state list
170                     state_list = []
171                     for i in range(len(final_prob_list)):
172                         state_list.append(i)
173
174                     final_prob_df = pd.DataFrame({'State':state_list,'Prob':final_prob_list})
175
176                     #if r % 10 == 0:
177                     #    print("DES rep: ",r," out of ",replications)
178
179                 ##### Calculate system stats #####
180                 ## expected value and state lists used in dataframe
181                 mean=0
182                 for i in range(len(final_prob_list)):
183                     mean += i*final_prob_list[i]
184                 ## variance
185                 var=0
186                 for i in range(len(final_prob_list)):
187                     var += (i-mean)*(i-mean)*final_prob_list[i]
188
189                 raw_diff = mean-overall_mean_numinsys
190                 abs_diff = abs(raw_diff)
191                 proportion_diff = raw_diff/overall_mean_numinsys
192                 stop_timer = timeit.default_timer()
193
194                 #append all of the lists to collect data in each iteration
195                 mean_list.append(mean)
196                 var_list.append(var)
197                 my_lambda_list.append(my_lambda)
198                 my_mu_list.append(my_mu)
199                 theoretical_mean_list.append(overall_mean_numinsys)
200                 theoretical_var_list.append(theoretical_var)
201                 raw_diff_list.append(raw_diff)
202                 abs_diff_list.append(abs_diff)
203                 proportion_diff_list.append(proportion_diff)
204                 state_probs_list.append(final_prob_list)
205                 run_time_list.append((stop_timer-start_timer)/60)
206
```

```
                print("DES loop",loop_counter)

        state_probs_df = pd.DataFrame(state_probs_list)
        DES_system_output = pd.DataFrame({"Mean":mean_list,"Theoretical_Mean":
    theoretical_mean_list,"Var":var_list,"Theoretical_Var":theoretical_var_list,"Lambda":
    my_lambda_list,"Mu":my_mu_list,"Raw Diff":raw_diff_list,"Abs Diff":abs_diff_list,"Run Time (
    min)":run_time_list,"Proportion Diff":proportion_diff_list})
        DES_system_output = pd.concat([DES_system_output,state_probs_df],axis=1)
        print("DES Complete!")


    if model_list[model] == "TOS":
        '''#############################################################################'''
        '''###########  Time-Oriented Simulation (TOS) - Markov Process ###########'''
        '''#############################################################################'''
        random.seed(53)
        mean_list = []
        var_list = []
        my_lambda_list = []
        my_mu_list = []
        theoretical_mean_list = []
        theoretical_var_list = []
        raw_diff_list = []
        abs_diff_list = []
        proportion_diff_list = []
        state_probs_list = []
        run_time_list = []

        loop_counter = 0
        for lambda_loop in range(len(lambda_list)):
            for mu_loop in range(len(mu_mult_scalar_list)):
                start_timer = timeit.default_timer()
                loop_counter += 1 #used to print loop counter while running code
                ### NOTE: lambda and mu are dependent on time interval size
                defined_lambda = lambda_list[lambda_loop]
                defined_mu = defined_lambda*mu_mult_scalar_list[mu_loop]
                my_lambda = defined_lambda*step_size
                my_mu = defined_mu*step_size
                my_rho = my_lambda/my_mu
                #overall_mean_theoretical = (my_rho*my_rho)/(1-my_rho)
                overall_mean_numinsys = my_rho/(1-my_rho)
                theoretical_var = my_rho/((1-my_rho)*(1-my_rho))

                #initialize time intervals (+1 added for indexing purposes but is not retained)
                if str(step_size)[::-1].find('.') == -1:
                    decimal_place = 0
                else:
                    decimal_place = str(step_size)[::-1].find('.')

                time_step_list = [x*step_size for x in range(0,round(time_end/step_size))]
                time_step_list = [round(x,decimal_place) for x in time_step_list] #round to amount
     of decimal places from step_size

                service_prob_list = []
```

```
                for arrival_loop in range(200):
                    poisson_temp_list = []
                    for i in range(200):
                        poisson_temp_list.append(poisson.pmf(i,(1/(arrival_loop+1))*my_mu))
                    service_prob_list.append(poisson_temp_list)
                service_prob_df = pd.DataFrame(service_prob_list)

                ### cdf
                service_cum_prob_list = []
                for row in range(len(service_prob_df)):
                    prob_list_cum = []
                    for col in range(len(service_prob_df.columns)): #first prob is just the pdf
                        if col == 0:
                            prob = service_prob_df.iloc[row,col]
                        elif col == len(service_prob_df.columns): #last prob is 1-sum of the
    others
                            prob = 1 - sum(service_prob_df.iloc[row,0:col-1])
                        else:
                            prob = service_prob_df.iloc[row,col] + sum(service_prob_df.iloc[row,0:
    col-1])
                        prob_list_cum.append(prob)
                    service_cum_prob_list.append(prob_list_cum)
                service_prob_cum_df = pd.DataFrame(service_cum_prob_list)

                row_names = []
                for i in range(len(service_prob_cum_df)):
                    row_names.append("A = %s" %i)
                col_names = []
                for i in range(len(service_prob_cum_df.columns)):
                    col_names.append("Serv = %s" %i)
                service_prob_cum_df.columns = col_names
                service_prob_cum_df.index = row_names
                service_prob_df.columns = col_names
                service_prob_df.index = row_names

                #initialize master matrix
                #This will store the probabilities lists as they are generated for each unique
    case
                temp_list = []
                for i in range(50): #creates nxn matrix
                    temp_list.append(0)

                row_names = []
                for i in range(len(temp_list)):
                    row_names.append("Ls = %s"%i)
                col_names = []
                for i in range(len(temp_list)):
                    col_names.append("A = %s"%i)
                master_prob_matrix_df = pd.DataFrame(columns=col_names,index=row_names)
                #chunk_prob_matrix = pd.DataFrame()
                for i in range(len(master_prob_matrix_df)):
                    for j in range(len(master_prob_matrix_df.columns)):
                        master_prob_matrix_df.iloc[i,j] = 0
                master_prob_matrix_df = master_prob_matrix_df.astype('object')
```

```
308
309             for r in range(replications): #amount of replications
310                 queue_length_list = []
311                 q_len = 0 #initialize queue length
312                 service_carryover = 0
313
314                 arrival_list = []
315                 cum_arrival = 0
316                 cum_arrival_list = []
317                 time_int_list = []
318
319                 for times in range(len(time_step_list)):
320                     rv_arr = random.uniform(0,1) #arrival
321                     rv_q_len = random.uniform(0,1) #service
322                     rv_decision = random.uniform(0,1) #determine if sevice generated prior to
    arrival
323
324                     finished=False
325                     arr_loop=0
326                     while not finished:
327                         if rv_arr <= poisson.cdf(arr_loop,my_lambda):
328                             arrival = arr_loop
329                             finished=True
330                         arr_loop+=1
331
332                     if master_prob_matrix_df.iloc[q_len,arrival] == 0:
333                         #initialize matrix for time chunks
334                         temp_list = []
335                         for i in range(30): #creates nxn matrix
336                             temp_list.append(0)
337
338                         row_names = []
339                         for i in range(len(temp_list)):
340                             row_names.append("Time Chunk: %s"%i)
341                         col_names = []
342                         for i in range(len(temp_list)):
343                             col_names.append("State = %s"%i)
344                         chunk_prob_matrix = pd.DataFrame(columns=col_names,index=row_names)
345                         #chunk_prob_matrix = pd.DataFrame()
346                         for i in range(len(chunk_prob_matrix)):
347                             for j in range(len(chunk_prob_matrix.columns)):
348                                 chunk_prob_matrix.iloc[i,j] = 0
349
350                         time_chunk = 0
351
352                         if q_len + arrival == 0:
353                             chunk_prob_matrix.iloc[0,0] = 1
354
355                         if q_len > 0 and arrival == 0: #hardcode row as 0 because arrival==0
356                             for state in range(q_len):
357                                 state +=1 #we want to index (1 to q_len)
358                                 chunk_prob_matrix.iloc[0,state] = service_prob_df.iloc[0,q_len
    -state]
359                             chunk_prob_matrix.iloc[0,0] = service_prob_df.iloc[0,q_len]
```

```
                                #chunk_prob_matrix.iloc[0,0] = 1 - sum(chunk_prob_matrix.iloc
    [0,:])

                                #### proportional spread
                                denom = sum(chunk_prob_matrix.iloc[0,:]) #denominator used for
    normalized proportion
                                residual_prob = 1 - denom

                                stop = 0
                                col = 0
                                while stop == 0:
                                    if chunk_prob_matrix.iloc[0,col] == 0 and chunk_prob_matrix.
    iloc[0,col+1] == 0:
                                        cutoff = col
                                        stop = 1
                                    col += 1

                                for res_prob_loop in range(cutoff):
                                    chunk_prob_matrix.iloc[0,res_prob_loop] += (chunk_prob_matrix.
    iloc[0,res_prob_loop]/denom)*residual_prob #add residual to existing prob proportionally

                            if arrival > 0:
                                chunk_prob_matrix.iloc[1,0] = 0

                                if q_len == 0:
                                    chunk_prob_matrix.iloc[1,1] = 1
                                elif q_len > 0:
                                    state = 2
                                    while state <= q_len+1:
                                        chunk_prob_matrix.iloc[1,state] = service_prob_df.iloc[
    arrival,q_len-state+1]
                                        state += 1
                                    chunk_prob_matrix.iloc[1,1] = service_prob_df.iloc[arrival,
    q_len]
                                    #chunk_prob_matrix.iloc[1,1] = 1- sum(chunk_prob_matrix.iloc
    [1,:])
                                    #### proportional spread
                                    denom = sum(chunk_prob_matrix.iloc[1,:]) #denominator used for
     normalized proportion
                                    residual_prob = 1 - denom

                                    stop = 0
                                    col = 0
                                    while stop == 0:
                                        if chunk_prob_matrix.iloc[1,col] == 0 and
    chunk_prob_matrix.iloc[1,col+1] == 0:
                                            cutoff = col
                                            stop = 1
                                        col += 1

                                    for res_prob_loop in range(cutoff):
                                        chunk_prob_matrix.iloc[1,res_prob_loop] += (
    chunk_prob_matrix.iloc[1,res_prob_loop]/denom)*residual_prob #add residual to existing prob
    proportionally
```

```
403
404                         if arrival != 0:
405                             time_chunk = 2
406                             while time_chunk <= arrival:
407                                 for state in range(1,q_len+time_chunk):
408                                     #state+=1 # fix zero index
409                                     for services in range(state):
410                                         chunk_prob_matrix.iloc[time_chunk,state-services+1] = chunk_prob_matrix.iloc[time_chunk,state-services+1] + chunk_prob_matrix.iloc[time_chunk-1,state]*service_prob_df.iloc[arrival,services]
411                                 chunk_prob_matrix.iloc[time_chunk,1] = service_prob_df.iloc[arrival,q_len]
412
413                                 #### proportional spread
414                                 denom = sum(chunk_prob_matrix.iloc[time_chunk,:]) #denominator used for normalized proportion
415                                 residual_prob = 1 - denom
416
417                                 stop = 0
418                                 col = 0
419                                 while stop == 0:
420                                     if chunk_prob_matrix.iloc[time_chunk,col] == 0 and chunk_prob_matrix.iloc[time_chunk,col+1] == 0:
421                                         cutoff = col
422                                         stop = 1
423                                     col += 1
424
425                                 for res_prob_loop in range(cutoff):
426                                     chunk_prob_matrix.iloc[time_chunk,res_prob_loop] += (chunk_prob_matrix.iloc[time_chunk,res_prob_loop]/denom)*residual_prob #add residual to existing prob proportionally
427
428                                 time_chunk += 1
429
430                             for state in range(q_len+arrival):
431                                 state += 1 #fix zero index
432                                 for services in range(state+1):
433                                     chunk_prob_matrix.iloc[arrival+1,state-services] = chunk_prob_matrix.iloc[arrival+1,state-services] + chunk_prob_matrix.iloc[arrival,state]*service_prob_df.iloc[arrival,services]
434
435                             for row in range(len(chunk_prob_matrix)):
436                                 chunk_prob_matrix.iloc[row,0] = 1-sum(chunk_prob_matrix.iloc[row,1:])
437
438                         stop = 0
439                         col = 0
440                         while stop == 0:
441                             if chunk_prob_matrix.iloc[time_chunk,col] == 0 and chunk_prob_matrix.iloc[time_chunk,col+1] == 0:
442                                 cutoff = col
443                                 stop = 1
444                             col += 1
445
```

```
                            prob_list = []
                            for i in range(cutoff):
                                prob_list.append(chunk_prob_matrix.iloc[time_chunk,i])


                            prob_list_cum = []
                            for i in range(len(prob_list)):
                                if i == 0:
                                    prob_list_cum.append(prob_list[i])
                                else:
                                    prob_list_cum.append(prob_list_cum[i-1]+prob_list[i])

                            master_prob_matrix_df.iloc[q_len,arrival] = prob_list_cum

                        prob_list_cum = master_prob_matrix_df.iloc[q_len,arrival]
                        '''generate services based on pseudo-random number'''
                        finished=False
                        j=0
                        while not finished:
                            if rv_q_len <= prob_list_cum[j]:
                                q_len = j
                                finished=True
                            j+=1

                        arrival_list.append(arrival)

                        queue_length_list.append(q_len)

                    arrival_mean = sum(arrival_list)/len(arrival_list)
                    ##combine queue length lists for each replication
                    queue_info_timestep = pd.DataFrame({'Time':time_step_list[0:len(time_step_list
)], 'Queue_Length %s' % r:queue_length_list[0:len(time_step_list)]})
                    queue_info_timestep = queue_info_timestep.iloc[n:]
                    #overall_queue_size_list_timestep.append(queue_size_list)
                    if r == 0:
                        overall_queue_size_timestep = queue_info_timestep
                    else:
                        overall_queue_size_timestep = pd.concat([overall_queue_size_timestep,
queue_info_timestep.iloc[:,1]],axis=1)

                    if r % 10 == 0:
                        print('TOS rep: ',r,' out of ',replications)

                overall_mean = overall_queue_size_timestep.loc[:,overall_queue_size_timestep.
columns != 'Time'].stack().mean()

                #create list to count the lengths to find the totals of each state
                count_lengths_list = []
                for i in range(50): #pick arbitrarily big number to account for all possibilities
                    count_lengths_list.append(0)
                #counts for each state
                for i in range(len(overall_queue_size_timestep)): #rows
                    for j in range(1,len(overall_queue_size_timestep.columns)): #cols
                        for lengths in range(len(count_lengths_list)):
```

```
                            if overall_queue_size_timestep.iloc[i,j] == lengths:
                                count_lengths_list[lengths] += 1
                #identify location where counts are not zero
                i=0
                while count_lengths_list[i] != 0:
                    i+=1
                #only keep counts that are not zero
                count_lengths_list = count_lengths_list[0:i]
                #list of states
                state_list = []
                for i in range(len(count_lengths_list)):
                    state_list.append(i)

                #calculate probabilities
                end_probs = []
                for i in range(len(count_lengths_list)):
                    end_probs.append(count_lengths_list[i]/sum(count_lengths_list))
                end_probs[-1]=1-sum(end_probs[0:len(end_probs)-1])

                ## expected value and state lists used in dataframe
                mean=0
                state_list = []
                for i in range(len(end_probs)):
                    mean += i*end_probs[i]
                    state_list.append(i)
                ## variance
                var=0
                for i in range(len(end_probs)):
                    var += (i-mean)*(i-mean)*end_probs[i]

                raw_diff = mean-overall_mean_numinsys
                abs_diff = abs(raw_diff)
                proportion_diff = raw_diff/overall_mean_numinsys
                stop_timer = timeit.default_timer()

                #append all of the lists to collect data in each iteration
                mean_list.append(mean)
                var_list.append(var)
                my_lambda_list.append(my_lambda)
                my_mu_list.append(my_mu)
                theoretical_mean_list.append(overall_mean_numinsys)
                theoretical_var_list.append(theoretical_var)
                raw_diff_list.append(raw_diff)
                abs_diff_list.append(abs_diff)
                proportion_diff_list.append(proportion_diff)
                state_probs_list.append(end_probs)
                run_time_list.append((stop_timer-start_timer)/60)

                print("TOS loop",loop_counter)
        state_probs_df = pd.DataFrame(state_probs_list)
        TOS_system_output = pd.DataFrame({"Mean":mean_list,"Theoretical_Mean":
    theoretical_mean_list,"Var":var_list,"Theoretical_Var":theoretical_var_list,"Lambda":
    my_lambda_list,"Mu":my_mu_list,"Raw Diff":raw_diff_list,"Abs Diff":abs_diff_list,"Run Time (
    min)":run_time_list,"Proportion Diff":proportion_diff_list})
```

```
        TOS_system_output = pd.concat([TOS_system_output,state_probs_df],axis=1)
        print("TOS Complete!")

    if model_list[model] == "DBN":
        '''#############################################################################'''
        '''##########  Dynamic Bayesian Network (DBN) - Markov Process  ##########'''
        '''#############################################################################'''
        print("DBN Started...")
        mean_list = []
        var_list = []
        my_lambda_list = []
        my_mu_list = []
        theoretical_mean_list = []
        theoretical_var_list = []
        raw_diff_list = []
        abs_diff_list = []
        proportion_diff_list = []
        state_probs_list = []
        run_time_list = []

        tol = 0.001

        loop_counter = 0
        for lambda_loop in range(len(lambda_list)):
            for mu_loop in range(len(mu_mult_scalar_list)):
                start_timer = timeit.default_timer()
                loop_counter += 1 #used to print loop counter while running code
                ### NOTE: lambda and mu are dependent on time interval size
                defined_lambda = lambda_list[lambda_loop]
                defined_mu = defined_lambda*mu_mult_scalar_list[mu_loop]
                my_lambda = defined_lambda*step_size
                my_mu = defined_mu*step_size
                my_rho = my_lambda/my_mu
                overall_mean_numinsys = my_rho/(1-my_rho)
                theoretical_var = my_rho/((1-my_rho)*(1-my_rho))

                arrival_cdf = []
                service_cdf = []
                arrival_pmf = []
                service_pmf = []

                ############  DEFINE PROB LIST LENGTHS  ############
                # create cdf probs for arrival and service
                for i in range(30):
                    arrival_cdf.append(poisson.cdf(i,my_lambda))
                    service_cdf.append(poisson.cdf(i,my_mu))
                #find location where change in prob is insignificant (important for arrivals)
                finished = False
                j=0
                while not finished:
                    if arrival_cdf[j+1] - arrival_cdf[j] <= tol:
                        arrival_cutoff = j
                        finished = True
                    j+=1
```

```
602
603                 finished = False
604                 j=0
605                 while not finished:
606                     if service_cdf[j+1] - service_cdf[j] <= tol:
607                         service_cutoff = j
608                         finished = True
609                     j+=1
610                 #overall_cutoff = max(arrival_cutoff,service_cutoff)
611
612                 ############ CREATE PDF LISTS (with truncation) ############
613                 #create pdf probs of equal length and last truncated due to tolerence level
614                 # arrivals - fixed for computations
615                 for i in range(arrival_cutoff+1):
616                     if i != arrival_cutoff:
617                         arrival_pmf.append(poisson.pmf(i,my_lambda))
618                     else:
619                         arrival_pmf.append(1-sum(arrival_pmf[0:arrival_cutoff+1]))
620
621                 ############   CALCULATIONS   ############
622                 length_multiplier = 2
623
624                 # initialize beginning state(empty and idle)
625                 start_probs = []
626                 for i in range(len(arrival_pmf)*length_multiplier):
627                     start_probs.append(0)
628                 start_probs[0] = 1 #assume empty and idle
629
630                 ### pdf
631                 service_prob_list = []
632                 for arrival_loop in range(200):
633                     poisson_temp_list = []
634                     for i in range(200):
635                         poisson_temp_list.append(poisson.pmf(i,(1/(arrival_loop+1))*my_mu))
636                     service_prob_list.append(poisson_temp_list)
637                 service_prob_df = pd.DataFrame(service_prob_list)
638
639                 ### cdf
640                 service_cum_prob_list = []
641                 for row in range(len(service_prob_df)):
642                     prob_list_cum = []
643                     for col in range(len(service_prob_df.columns)): #first prob is just the pdf
644                         if col == 0:
645                             prob = service_prob_df.iloc[row,col]
646                         elif col == len(service_prob_df.columns): #last prob is 1-sum of the
    others
647                             prob = 1 - sum(service_prob_df.iloc[row,0:col-1])
648                         else:
649                             prob = service_prob_df.iloc[row,col] + sum(service_prob_df.iloc[row,0:
    col-1])
650                         prob_list_cum.append(prob)
651                     service_cum_prob_list.append(prob_list_cum)
652                 service_prob_cum_df = pd.DataFrame(service_cum_prob_list)
653
```

```
654                 #initialize master matrix. This will store the probabilities lists as they are
    generated for each unique case
655                 temp_list = []
656                 for i in range(50): #creates nxn matrix
657                     temp_list.append(0)
658 
659                 row_names = []
660                 for i in range(len(temp_list)):
661                     row_names.append("Ls = %s"%i)
662                 col_names = []
663                 for i in range(len(temp_list)):
664                     col_names.append("A = %s"%i)
665                 master_prob_matrix_df = pd.DataFrame(columns=col_names,index=row_names)
666                 #chunk_prob_matrix = pd.DataFrame()
667                 for i in range(len(master_prob_matrix_df)):
668                     for j in range(len(master_prob_matrix_df.columns)):
669                         master_prob_matrix_df.iloc[i,j] = 0
670                 master_prob_matrix_df = master_prob_matrix_df.astype('object')
671 
672                 # calculations for each time step. Stop when state probs converge based on tol
    level
673                 check=0 #this stops the loop if
674                 while check != len(start_probs):
675                     end_probs = []
676                     for reset in range(len(start_probs)):
677                         end_probs.append(0)
678                     # start + arrival - service = end
679                     for end_state in range(len(end_probs)):
680                         for start_state in range(len(start_probs)):
681                             for arrivals in range(len(arrival_pmf)):
682                                 #for services in range(arrival_cutoff+10):
683                                 for services in range(start_state+arrivals+1):
684                                     if start_state + arrivals - services == end_state: #only
    feasible combinations
685                                         if master_prob_matrix_df.iloc[start_state,arrivals] == 0:
686                                             #initialize matrix for time chunks
687                                             temp_list = []
688                                             for i in range(len(end_probs)+20): #creates matrix
    bigger than end_probs list
689                                                 temp_list.append(0)
690 
691                                             row_names = []
692                                             for i in range(len(temp_list)):
693                                                 row_names.append("Time Chunk: %s"%i)
694                                             col_names = []
695                                             for i in range(len(temp_list)):
696                                                 col_names.append("State = %s"%i)
697 
698                                             chunk_prob_matrix = pd.DataFrame(columns=col_names,
    index=row_names)
699                                             for i in range(len(chunk_prob_matrix)):
700                                                 for j in range(len(chunk_prob_matrix.columns)):
701                                                     chunk_prob_matrix.iloc[i,j] = 0
702 
```

```
                                        time_chunk = 0

                                        if start_state + arrivals == 0:
                                            chunk_prob_matrix.iloc[0,0] = 1

                                        if start_state > 0 and arrivals == 0:
                                            for state in range(start_state):
                                                state +=1
                                                chunk_prob_matrix.iloc[0,state] =
    service_prob_df.iloc[0,start_state-state]
                                            chunk_prob_matrix.iloc[0,0] = service_prob_df.iloc
    [0,start_state]
                                            #### proportional spread
                                            denom = sum(chunk_prob_matrix.iloc[0,:]) #
    denominator used for normalized proportion
                                            residual_prob = 1 - denom

                                            stop = 0
                                            col = 0
                                            while stop == 0:
                                                if chunk_prob_matrix.iloc[0,col] == 0 and
    chunk_prob_matrix.iloc[0,col+1] == 0:
                                                    cutoff = col
                                                    stop = 1
                                                col += 1

                                            for res_prob_loop in range(cutoff):
                                                chunk_prob_matrix.iloc[0,res_prob_loop] += (
    chunk_prob_matrix.iloc[0,res_prob_loop]/denom)*residual_prob #add residual to existing prob
    proportionally

                                        if arrivals > 0:
                                            chunk_prob_matrix.iloc[1,0] = 0

                                            if start_state == 0:
                                                chunk_prob_matrix.iloc[1,1] = 1
                                            elif start_state > 0:
                                                state = 2
                                                while state <= start_state+1:
                                                    chunk_prob_matrix.iloc[1,state] =
    service_prob_df.iloc[arrivals,start_state-state+1]
                                                    state += 1
                                                chunk_prob_matrix.iloc[1,1] = service_prob_df.
    iloc[arrivals,start_state]
                                                #### proportional spread
                                                denom = sum(chunk_prob_matrix.iloc[1,:]) #
    denominator used for normalized proportion
                                                residual_prob = 1 - denom

                                                stop = 0
                                                col = 0
                                                while stop == 0:
                                                    if chunk_prob_matrix.iloc[1,col] == 0 and
    chunk_prob_matrix.iloc[1,col+1] == 0:
```

```
                                                    cutoff = col
                                                    stop = 1
                                                col += 1

                                            for res_prob_loop in range(cutoff):
                                                chunk_prob_matrix.iloc[1,res_prob_loop] += (chunk_prob_matrix.iloc[1,res_prob_loop]/denom)*residual_prob #add residual to existing prob proportionally

                                        if arrivals != 0:
                                            time_chunk = 2

                                            while time_chunk <= arrivals:
                                                for state in range(1,start_state+time_chunk):
                                                    #state+=1 # fix zero index
                                                    for serv in range(state):
                                                        chunk_prob_matrix.iloc[time_chunk,state-serv+1] = chunk_prob_matrix.iloc[time_chunk,state-serv+1] + chunk_prob_matrix.iloc[time_chunk-1,state]*service_prob_df.iloc[arrivals,serv]
                                                chunk_prob_matrix.iloc[time_chunk,1] = service_prob_df.iloc[arrivals,start_state]
                                                #### proportional spread
                                                denom = sum(chunk_prob_matrix.iloc[time_chunk,:]) #denominator used for normalized proportion
                                                residual_prob = 1 - denom

                                                stop = 0
                                                col = 0
                                                while stop == 0:
                                                    if chunk_prob_matrix.iloc[time_chunk,col] == 0 and chunk_prob_matrix.iloc[time_chunk,col+1] == 0:
                                                        cutoff = col
                                                        stop = 1
                                                    col += 1

                                                for res_prob_loop in range(cutoff):
                                                    chunk_prob_matrix.iloc[time_chunk,res_prob_loop] += (chunk_prob_matrix.iloc[time_chunk,res_prob_loop]/denom)*residual_prob #add residual to existing prob proportionally

                                                time_chunk += 1

                                            for state in range(start_state+arrivals):
                                                state += 1 #fix zero index
                                                for serv in range(state+1):
                                                    chunk_prob_matrix.iloc[arrivals+1,state-serv] = chunk_prob_matrix.iloc[arrivals+1,state-serv] + chunk_prob_matrix.iloc[arrivals,state]*service_prob_df.iloc[arrivals,serv]
                                            for row in range(len(chunk_prob_matrix)):
                                                chunk_prob_matrix.iloc[row,0] = 1-sum(chunk_prob_matrix.iloc[row,1:])

                                        #find stopping point of probabilities
                                        stop = 0
```

```
                                        col = 0
                                        while stop == 0:
                                            if chunk_prob_matrix.iloc[time_chunk,col] == 0 and chunk_prob_matrix.iloc[time_chunk,col+1] == 0:
                                                cutoff = col
                                                stop = 1
                                            col += 1

                                        prob_list = []
                                        for i in range(cutoff):
                                            prob_list.append(chunk_prob_matrix.iloc[time_chunk,i])

                                        prob_list_cum = []
                                        for i in range(len(prob_list)):
                                            if i == 0:
                                                prob_list_cum.append(prob_list[i])
                                            else:
                                                prob_list_cum.append(prob_list_cum[i-1]+prob_list[i])

                                        master_prob_matrix_df.iloc[start_state,arrivals] = prob_list

                                    prob_list = master_prob_matrix_df.iloc[start_state,arrivals]
                                    end_probs[end_state] += start_probs[start_state]*arrival_pmf[arrivals]*prob_list[end_state]

                end_probs[-1]=1-sum(end_probs[0:len(end_probs)-1])

                #stop of all of the state probs converge within tolerence level
                check=0
                for j in range(len(start_probs)):
                    if abs(start_probs[j]-end_probs[j]) <= conv_tol:
                        check += 1

                start_probs = end_probs

            ## expected value and state lists used in dataframe
            mean=0
            state_list = []
            for i in range(len(end_probs)):
                mean += i*end_probs[i]
                state_list.append(i)
            #print("Mean: ",mean)
            ## variance
            var=0
            for i in range(len(end_probs)):
                var += (i-mean)*(i-mean)*end_probs[i]
            #print("Variance: ",var)

            raw_diff = mean-overall_mean_numinsys
            abs_diff = abs(raw_diff)
```

```
                proportion_diff = raw_diff/overall_mean_numinsys
                stop_timer = timeit.default_timer()

                #append all of the lists to collect data in each iteration
                mean_list.append(mean)
                var_list.append(var)
                my_lambda_list.append(my_lambda)
                my_mu_list.append(my_mu)
                theoretical_mean_list.append(overall_mean_numinsys)
                theoretical_var_list.append(theoretical_var)
                raw_diff_list.append(raw_diff)
                abs_diff_list.append(abs_diff)
                proportion_diff_list.append(proportion_diff)
                state_probs_list.append(end_probs)
                run_time_list.append((stop_timer-start_timer)/60)

                print("DBN loop ",loop_counter)

        state_probs_df = pd.DataFrame(state_probs_list)
        DBN_system_output = pd.DataFrame({"Mean":mean_list,"Theoretical_Mean":
    theoretical_mean_list,"Var":var_list,"Theoretical_Var":theoretical_var_list,"Lambda":
    my_lambda_list,"Mu":my_mu_list,"Raw Diff":raw_diff_list,"Abs Diff":abs_diff_list,"Run Time (
    min)":run_time_list,"Proportion Diff":proportion_diff_list})
        DBN_system_output = pd.concat([DBN_system_output,state_probs_df],axis=1)
        print("DBN Complete!")

######################################################################
#################### Final Output Preparation ####################
######################################################################

### sort and join all of the output files
master_output = pd.DataFrame()
if model_list == ["DES","TOS","DBN"]:
    for row in range(max(len(DES_system_output),len(TOS_system_output),len(DBN_system_output))):
        master_output = pd.concat([master_output,DES_system_output.iloc[row,:],TOS_system_output.
    iloc[row,:],DBN_system_output.iloc[row,:]],axis=1,sort=False)
elif model_list == ["DES","TOS"]:
    for row in range(max(len(DES_system_output),len(TOS_system_output))):
        master_output = pd.concat([master_output,DES_system_output.iloc[row,:],TOS_system_output.
    iloc[row,:]],axis=1,sort=False)
elif model_list == ["DES","DBN"]:
    for row in range(max(len(DES_system_output),len(DBN_system_output))):
        master_output = pd.concat([master_output,DES_system_output.iloc[row,:],DBN_system_output.
    iloc[row,:]],axis=1,sort=False)
elif model_list == ["TOS","DBN"]:
    for row in range(max(len(TOS_system_output),len(DBN_system_output))):
        master_output = pd.concat([master_output,TOS_system_output.iloc[row,:],DBN_system_output.
    iloc[row,:]],axis=1,sort=False)
elif model_list == ["DES"]:
    for row in range(len(DES_system_output)):
        master_output = pd.concat([master_output,DES_system_output.iloc[row,:]],axis=1,sort=False)
elif model_list == ["TOS"]:
    for row in range(len(TOS_system_output)):
        master_output = pd.concat([master_output,TOS_system_output.iloc[row,:]],axis=1,sort=False)
```

```
elif model_list == ["DBN"]:
    for row in range(len(DBN_system_output)):
        master_output = pd.concat([master_output,DBN_system_output.iloc[row,:]],axis=1,sort=False)

### column labels for final output
name_list = []
for i in range(round(len(master_output.columns)/len(model_list))):
    for check in range(len(model_list)):
        if model_list[check] == "DES":
            name_list.append('DES')
        if model_list[check] == "TOS":
            name_list.append('TOS')
        if model_list[check] == "DBN":
            name_list.append('DBN')
master_output.columns = name_list

### replace nan's with 0 (nan's happen because of varying state lengths between reps)
for row in range(len(master_output)):
    for col in range(len(master_output.columns)):
        if np.isnan(master_output.iloc[row,col]) == True:
            master_output.iloc[row,col] = 0

#output to excel
if output_decision == 'Yes':
    master_output.to_excel(outfile)
```

Appendix B. Model Output

Table 7. DES Model Output

Model	Mean	Theoretical Mean	Variance	Theoretical Variance	Lambda	Mu	Run Time (min)	States																			
								0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
DES	1.003	1.000	2.007	2.000	0.1	0.2	0.113	0.501	0.247	0.125	0.063	0.032	0.015	0.008	0.004	0.002	0.001	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.676	0.667	1.128	1.111	0.1	0.3	0.123	0.597	0.240	0.097	0.041	0.016	0.006	0.003	0.001	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.500	0.500	0.745	0.750	0.1	0.3	0.106	0.666	0.222	0.076	0.024	0.008	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.333	0.333	0.438	0.444	0.1	0.4	0.140	0.750	0.187	0.048	0.012	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	1.001	1.000	1.968	2.000	0.3	0.5	0.150	0.498	0.250	0.127	0.063	0.031	0.015	0.007	0.004	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.669	0.667	1.115	1.111	0.3	0.6	0.109	0.600	0.240	0.096	0.039	0.016	0.006	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.496	0.500	0.726	0.750	0.3	0.8	0.131	0.667	0.223	0.074	0.025	0.008	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.335	0.333	0.446	0.444	0.3	1.0	0.149	0.750	0.186	0.048	0.012	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.992	1.000	1.985	2.000	0.5	1.0	0.129	0.502	0.250	0.125	0.061	0.031	0.015	0.008	0.004	0.002	0.001	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.672	0.667	1.132	1.111	0.5	1.3	0.160	0.600	0.239	0.096	0.040	0.016	0.006	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.503	0.500	0.762	0.750	0.5	1.5	0.112	0.667	0.220	0.075	0.026	0.008	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.335	0.333	0.444	0.444	0.5	2.0	0.120	0.748	0.189	0.047	0.012	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.989	1.000	1.915	2.000	0.8	1.5	0.111	0.501	0.249	0.126	0.063	0.031	0.015	0.008	0.004	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.673	0.667	1.142	1.111	0.8	1.9	0.147	0.598	0.240	0.096	0.039	0.015	0.007	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.497	0.500	0.744	0.750	0.8	2.3	0.111	0.668	0.222	0.074	0.024	0.008	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.332	0.333	0.443	0.444	0.8	3.0	0.110	0.751	0.186	0.046	0.012	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	1.005	1.000	1.973	2.000	0.9	1.8	0.113	0.496	0.251	0.127	0.064	0.032	0.016	0.007	0.004	0.002	0.001	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.677	0.667	1.135	1.111	0.9	2.3	0.164	0.597	0.241	0.096	0.040	0.016	0.007	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.495	0.500	0.744	0.750	0.9	2.7	0.148	0.669	0.221	0.073	0.025	0.008	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.336	0.333	0.433	0.444	0.9	3.6	0.133	0.750	0.187	0.047	0.012	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.981	1.000	1.934	2.000	1.0	2.0	0.113	0.503	0.252	0.125	0.061	0.030	0.015	0.007	0.004	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.675	0.667	1.108	1.111	1.0	2.5	0.134	0.595	0.242	0.096	0.040	0.016	0.006	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.499	0.500	0.739	0.750	1.0	3.0	0.142	0.667	0.221	0.075	0.025	0.008	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.329	0.333	0.440	0.444	1.0	4.0	0.123	0.752	0.186	0.046	0.012	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	1.002	1.000	2.017	2.000	1.5	3.0	0.115	0.502	0.248	0.124	0.063	0.031	0.016	0.008	0.004	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.668	0.667	1.093	1.111	1.5	3.8	0.146	0.596	0.243	0.097	0.038	0.015	0.006	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.492	0.500	0.729	0.750	1.5	4.5	0.115	0.670	0.221	0.073	0.025	0.008	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.334	0.333	0.451	0.444	1.5	6.0	0.132	0.751	0.186	0.047	0.012	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.992	1.000	1.949	2.000	2.0	4.0	0.113	0.500	0.251	0.125	0.062	0.031	0.016	0.007	0.004	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.674	0.667	1.129	1.111	2.0	5.0	0.111	0.598	0.240	0.096	0.040	0.015	0.006	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.493	0.500	0.735	0.750	2.0	6.0	0.111	0.669	0.223	0.073	0.024	0.008	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DES	0.334	0.333	0.447	0.444	2.0	8.0	0.112	0.750	0.187	0.047	0.012	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table 8. TOS Model Output

								States															
Model	Mean	Theoretical Mean	Variance	Theoretical Variance	Lambda	Mu	Run Time (min)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
TOS	1.053	1.000	1.996	2.000	0.1	0.2	12.234	0.472	0.263	0.134	0.067	0.034	0.017	0.008	0.004	0.002	0.001	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.729	0.667	1.225	1.111	0.1	0.3	11.910	0.569	0.258	0.103	0.040	0.018	0.009	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.517	0.500	0.758	0.750	0.1	0.3	11.653	0.650	0.240	0.073	0.024	0.008	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.372	0.333	0.490	0.444	0.1	0.4	11.651	0.724	0.206	0.051	0.014	0.004	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	1.014	1.000	1.934	2.000	0.3	0.5	5.030	0.476	0.273	0.128	0.065	0.031	0.014	0.006	0.003	0.00[illegible]	0.001	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.692	0.667	1.064	1.111	0.3	0.6	5.244	0.568	0.273	0.099	0.038	0.014	0.006	0.002	0.001	0.00[illegible]	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.541	0.500	0.775	0.750	0.3	0.8	4.880	0.634	0.249	0.078	0.027	0.009	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.359	0.333	0.437	0.444	0.3	1.0	5.186	0.721	0.219	0.047	0.010	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.981	1.000	1.913	2.000	0.5	1.0	2.701	0.486	0.280	0.120	0.057	0.028	0.014	0.007	0.004	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.673	0.667	1.068	1.111	0.5	1.3	2.936	0.579	0.267	0.096	0.034	0.014	0.006	0.002	0.001	0.00[illegible]	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.510	0.500	0.702	0.750	0.5	1.5	2.641	0.646	0.246	0.076	0.022	0.007	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.342	0.333	0.447	0.444	0.5	2.0	2.861	0.741	0.196	0.049	0.010	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.971	1.000	1.808	2.000	0.8	1.5	2.203	0.488	0.273	0.126	0.058	0.028	0.014	0.006	0.003	0.00[illegible]	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.661	0.667	1.062	1.111	0.8	1.9	2.300	0.590	0.257	0.094	0.035	0.014	0.006	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.490	0.500	0.699	0.750	0.8	2.3	2.196	0.665	0.228	0.076	0.021	0.007	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.320	0.333	0.429	0.444	0.8	3.0	2.005	0.762	0.174	0.050	0.011	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.957	1.000	1.830	2.000	0.9	1.8	1.893	0.498	0.270	0.122	0.055	0.027	0.013	0.007	0.004	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.654	0.667	1.088	1.111	0.9	2.3	1.796	0.601	0.243	0.097	0.035	0.013	0.005	0.003	0.001	0.00[illegible]	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.480	0.500	0.701	0.750	0.9	2.7	1.911	0.676	0.215	0.076	0.023	0.007	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.310	0.333	0.421	0.444	0.9	3.6	1.669	0.772	0.164	0.051	0.010	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.961	1.000	1.870	2.000	1.0	2.0	1.634	0.502	0.260	0.125	0.055	0.029	0.013	0.007	0.004	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.628	0.667	0.986	1.111	1.0	2.5	1.570	0.612	0.237	0.097	0.034	0.013	0.005	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.472	0.500	0.703	0.750	1.0	3.0	1.586	0.686	0.205	0.076	0.023	0.007	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.312	0.333	0.441	0.444	1.0	4.0	1.539	0.776	0.157	0.052	0.012	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.898	1.000	1.718	2.000	1.5	3.0	1.594	0.530	0.241	0.125	0.055	0.026	0.012	0.006	0.002	0.00[illegible]	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.622	0.667	1.058	1.111	1.5	3.8	1.362	0.631	0.212	0.099	0.035	0.014	0.005	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.457	0.500	0.707	0.750	1.5	4.5	1.282	0.703	0.185	0.078	0.024	0.007	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.303	0.333	0.477	0.444	1.5	6.0	1.292	0.796	0.131	0.054	0.015	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.888	1.000	1.692	2.000	2.0	4.0	1.281	0.538	0.231	0.126	0.056	0.025	0.012	0.006	0.002	0.00[illegible]	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.634	0.667	1.112	1.111	2.0	5.0	1.189	0.635	0.199	0.100	0.040	0.015	0.005	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.464	0.500	0.761	0.750	2.0	6.0	1.180	0.712	0.167	0.082	0.027	0.008	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TOS	0.303	0.333	0.495	0.444	2.0	8.0	1.119	0.802	0.121	0.055	0.016	0.004	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table 9. DBN Model Output

								States																
Model	Mean	Theoretical Mean	Variance	Theoretical Variance	Lambda	Mu	Run Time (min)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
DBN	0.915	1.000	1.401	2.000	0.1	0.2	0.189	0.495	0.268	0.128	0.063	0.031	0.016	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.658	0.667	0.952	1.111	0.1	0.3	0.190	0.586	0.258	0.097	0.038	0.015	0.006	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.511	0.500	0.699	0.750	0.1	0.3	0.196	0.648	0.241	0.075	0.025	0.008	0.003	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.354	0.333	0.443	0.444	0.1	0.4	0.184	0.729	0.208	0.048	0.012	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.986	1.000	1.702	2.000	0.3	0.5	0.227	0.478	0.277	0.127	0.062	0.031	0.015	0.007	0.004	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.687	0.667	1.043	1.111	0.3	0.6	0.233	0.572	0.269	0.098	0.038	0.015	0.006	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.528	0.500	0.730	0.750	0.3	0.8	0.227	0.636	0.252	0.076	0.024	0.008	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.362	0.333	0.450	0.444	0.3	1.0	0.228	0.721	0.215	0.049	0.011	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.986	1.000	1.784	2.000	0.5	1.0	0.321	0.478	0.281	0.125	0.059	0.029	0.014	0.007	0.004	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.675	0.667	1.036	1.111	0.5	1.3	0.296	0.576	0.269	0.097	0.036	0.014	0.006	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.513	0.500	0.715	0.750	0.5	1.5	0.322	0.645	0.246	0.075	0.023	0.007	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.343	0.333	0.437	0.444	0.5	2.0	0.327	0.739	0.199	0.049	0.011	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.955	1.000	1.727	2.000	0.8	1.5	0.306	0.490	0.276	0.124	0.056	0.027	0.014	0.007	0.003	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.651	0.667	1.006	1.111	0.8	1.9	0.297	0.592	0.256	0.096	0.034	0.013	0.005	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.491	0.500	0.699	0.750	0.8	2.3	0.302	0.665	0.228	0.075	0.022	0.007	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.321	0.333	0.431	0.444	0.8	3.0	0.300	0.761	0.174	0.050	0.011	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.946	1.000	1.763	2.000	0.9	1.8	0.407	0.498	0.270	0.124	0.056	0.027	0.013	0.007	0.003	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.640	0.667	1.011	1.111	0.9	2.3	0.405	0.602	0.246	0.096	0.034	0.013	0.005	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.481	0.500	0.703	0.750	0.9	2.7	0.411	0.675	0.216	0.076	0.022	0.007	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.313	0.333	0.437	0.444	0.9	3.6	0.405	0.772	0.162	0.051	0.012	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.936	1.000	1.751	2.000	1.0	2.0	0.429	0.504	0.265	0.124	0.055	0.026	0.013	0.006	0.003	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.633	0.667	1.008	1.111	1.0	2.5	0.426	0.608	0.240	0.097	0.034	0.013	0.005	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.476	0.500	0.705	0.750	1.0	3.0	0.435	0.682	0.209	0.077	0.023	0.007	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.309	0.333	0.440	0.444	1.0	4.0	0.426	0.778	0.154	0.052	0.012	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.909	1.000	1.752	2.000	1.5	3.0	0.606	0.527	0.243	0.125	0.055	0.026	0.013	0.006	0.003	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.618	0.667	1.031	1.111	1.5	3.8	0.604	0.630	0.213	0.099	0.036	0.013	0.005	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.466	0.500	0.736	0.750	1.5	4.5	0.608	0.702	0.182	0.080	0.025	0.008	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.302	0.333	0.468	0.444	1.5	6.0	0.627	0.795	0.132	0.054	0.015	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.907	1.000	1.787	2.000	2.0	4.0	0.863	0.538	0.227	0.126	0.058	0.026	0.013	0.006	0.003	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.623	0.667	1.078	1.111	2.0	5.0	0.918	0.639	0.197	0.101	0.039	0.014	0.005	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.472	0.500	0.783	0.750	2.0	6.0	0.979	0.709	0.168	0.082	0.028	0.009	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DBN	0.307	0.333	0.501	0.444	2.0	8.0	0.929	0.800	0.121	0.056	0.017	0.004	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Bibliography

1. MIT OpenCouseware Chapter 2 Poisson Processes, Theorem 2.5.1. `https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-262-discrete-stochastic-processes-spring-2011/course-notes/MIT6_262S11_chap02.pdf`. Accessed: 2020-02-26.

2. J. Banks, J. Carson, B. Nelson, and D. Nicol. *Discrete-Event System Simulation*. Fourth edition.

3. A. Buss and A. A. Rowaei. A COMPARISON OF THE ACCURACY OF DISCRETE EVENT AND DISCRETE TIME. pages 1468–1477, 2010.

4. R. W. Conway and B. M. Johnson. DELAY TIMES IN AN M/M/1 QUEUE: ESTIMATING THE SAMPLING DISTRIBUTIONS FOR THE STEADY-STATE MEAN AND MSER TRUNCATION POINT. 1959.

5. C. M. Harris. Markov chains. (1):481–485, 1986.

6. M. Jacyna, J. Żak, and P. Gołębiowski. The Use of the Queueing Theory for the Analysis of Transport Processes. pages 101–112, 2015.

7. R. Keeney and D. von Winterfeldt. Complex Technical Problems. *IEEE Transactions on Engineering Management*, 38(3):191–201, 1991.

8. C. T. Kelleher. *Dynamic Bayesian Networks as a Probabalistic Metamodel for Combat Simulation*. PhD thesis, Air Force Institute of Technology, 2014.

9. S. C. D. Little, R. Casas-mulet, L. Patulny, J. Wand, K. A. Miller, F. Fidler, M. J. Stewardson, and J. A. Webb. Environmental Modelling & Software Minimising biases in expert elicitations to inform environmental management : Case studies from environmental fl ows in Australia. *Environmental Modelling and Software*, 100:146–158, 2018.

10. X. Ma. Bayesian networks-based data publishing method using smooth sensitivity. *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, pages 795–800, 2018.

11. H. Mayfield, E. Bertone, O. Sahin, and C. Smith. Structurally Aware Discretisation for Bayesian Networks. Technical Report December, 2017.

12. R. E. Neapolitan. *Learning Bayesian Networks*. Pearson Prentice Hall, first edition, 2004.

13. J. Poropudas and K. Virtanen. Analyzing Air Combat Simulation Results with Dynamic Bayesian Networks. In *Proceedings of the 39th conference on Winter simulation: 40 years! The best is yet to come*, pages 1370–1377. IEEE Press, 2007.

14. S. M. Ross. *Introduction to Probability Models*. Elsevier Inc., eleventh edition, 2014.

15. A. Saksena and R. Saunders.

16. R. Saunders and A. Saksena.

17. J. X. Situ. Combat Identification of Synthetic Aperture Radar Images using Contextual Features and Bayesian Belief Networks. Master's thesis, Air Force Institute of Technology, 2012.

18. T. A. Stephenson. An Introduction to Bayesian Network Theory and Usage. Technical report, Dalle Molle Institute for Perceptual Artificial Intelligence, 2000.

19. X. Tian and K. Benkrid. Mersenne Twister Random Number Generation on FPGA , CPU and GPU. *2009 NASA/ESA Conference on Adaptive Hardware and Systems*, pages 460–464, 2009.

20. A. Veiga and E. Spinelli. A pulse generator with poisson-exponential distribution for emulation of radioactive decay events. In *2016 IEEE 7th Latin American Symposium on Circuits & Systems (LASCAS)*, pages 31–34. IEEE, 2016.

21. K. P. White and S. N. Hwang. DELAY TIMES IN AN M/M/1 QUEUE: ESTIMATING THE SAMPLING DISTRIBUTIONS FOR THE STEADY-STATE MEAN AND MSER TRUNCATION POINT. *2015 Winter Simulation Conference (WSC)*, (2010):493–504, 2015.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
26-03-2020	Master's Thesis	SEP 2018 - MAR 2020

4. TITLE AND SUBTITLE

Analysis with Bayesian Networks Compared to Simulation

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Salazar, Aaron, J. Capt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENS-MS-20-168

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Intentionally left blank

10. SPONSOR/MONITOR'S ACRONYM(S)

Intentionally left blank

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This research compares simulations to Dynamic Bayesian Networks in analyzing situations. The research applies models that have known output mean and variance. Queueing systems have theoretical values of the steady-state mean and variance for the number of entities in the system. Monte Carlo simulation development is broken down into two separate approaches: discrete-event simulation and time-oriented simulation. The discrete-event simulation uses pseudo-random numbers to schedule and trigger future events (i.e. customer arrivals and services) and is based on the generated objects. The time-oriented simulation utilizes fixed-width time intervals and updates the system state according to a stochastic process for the set of events occurring during each time period. The accuracy of each approach in estimated by a comparison to the theoretical mean, variance, and probability values.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Mark A. Gallagher, Ph.D., AFIT/ENS
U	U	U	UU	81	19b. TELEPHONE NUMBER *(include area code)* 785-3636, x4703; mark.gallagher@afit.edu