5-2009

Tandem Queues with Non-Stationary Arrivals

Senthil balaji Girimurugan
Clemson University, coolbbeejay@gmail.com

Tandem Queues with Non-Stationary Arrivals

A Thesis

Presented to

the Graduate School of

Clemson University

In Partial Fulfillment

of the Requirements for the Degree

Master of Sciences

Mathematical Sciences

---

by

Senthil Balaji Girimurugan

May 2009

Advisor: Dr.Peter C. Kiessler

Co-Advisor(s): Dr.Robert B. Lund , Dr.Colin M. Gallagher

ABSTRACT

We consider a network of K queues in tandem labeled $Q_1, Q_2, .., Q_K$. The arrivals to Q_1 form a non-homogeneous Poisson process whose intensity is periodic. We conjecture that asymptotically the arrival process A_j to $Q_j, j = 1, 2, .., K$ is cycle stationary. In addition, we conjecture that asymptotically as j gets larger, the arrival process at the j^{th} queue gets closer to a stationary point process. Hence, the queue performance measures become more stationary as j increases. We perform Monte-Carlo simulations and design statistical tests whose results support the conjecture.

To my dear wife Lela and family

ACKNOWLEDGMENTS

I am very grateful to my advisors who have provided a great deal of support in my research. They have been helpful not only in my research but also in offering a strong knowledge base through valuable coursework that lead to this research work. I am indebted greatly to Dr. Peter C. Kiessler who has been the mainstay of my academic progress, taking the time to correct my mistakes and hone my skills in Statistics and Stochastics.

TABLE OF CONTENTS

Table of Contents (Continued)

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION TO NON-HOMOGENEOUS TANDEM QUEUES

We consider a system consisting of K queues in series labeled $Q_1, Q_2, .., Q_K$. The arrival process $\{A(t);\ t \geq 0\}$ to Q_1 is a non-homogeneous Poisson process with intensity $\{\lambda(t); t \geq 0\}$. The service times $S_{k,n},\ k = 1, 2, .., K,\ n = 1, 2, \ldots$ are independent with distribution function F_k and mean μ_k. The arrival proces $\{A(t);\ t \geq 0\}$ and the K service time sequences $\{S_1\}, \{S_2\}.., \{S_K\}$ are assumed independent.

An example that mimics our design is a simple cafeteria model. Consider a cafeteria with 5 different food service stations. A customer has the option of going through all the stations or only a couple of them before reaching the check out. Thus, each station can be visualized as a single server queue in a queueing system. The system in our case is employed under the assumption that the customer goes through all the stations in series before reaching the check out counter. The reason behind this assumption is merely to attain simplicity in the simulation. Like a cafeteria that operates under a specified time period, the traffic intensity in our design is defined over a restricted time interval (0,D). Under these restrictions, the service times at each queue has to be chosen appropriately in order to avoid traffic overload. Thus, the mean service time is chosen in such a manner that the following condition holds

$$\Lambda(D) = \frac{1}{D}\int_0^D \lambda(t)dt \leq \min(\mu_1, \mu_2, \ldots \mu_K). \tag{1.1}$$

1.1 Non-Homogeneous Poisson Process

We assume that the arrival process to Q_1 is a non-homogeneous Poisson process where the intensity function is periodic. That is, $\{A(t), t \geq 0\}$ is a non-homogeneous Poisson process with intensity function $\{\lambda(t), t \geq 0\}$ having period 'D'. This can be expressed as,

$$\lambda(t + D) = \lambda(t), \forall t \geq 0. \tag{1.2}$$

Thus, $\{A(t), t \geq 0\}$ has independent increments and are cycle stationary in the following sense that[1] ,

$$A(t+s) - A(t) \stackrel{\circ}{=} A(D+t+s) - A(D+t). \tag{1.3}$$

A detailed description of one such process is discussed in the next chapter.

1.2 Design of the Tandem Queue System

The tandem queue design consists of 'K' single server queues defined as $Q_1, Q_2 \ldots Q_K$. The arrival processes to each of the queues are given by $A_1, A_2, \ldots A_K$ and the service distributions are given by $S_1, S_2, \ldots S_K$. Our design involves 10 nodes as shown in the figure below. However, the first node is used to generate arrivals and the last node accounts for departures . They are depicted for the purposes of better understanding. Thus, there are strictly eight server nodes of interest in our design as shown in Figure 1.1.

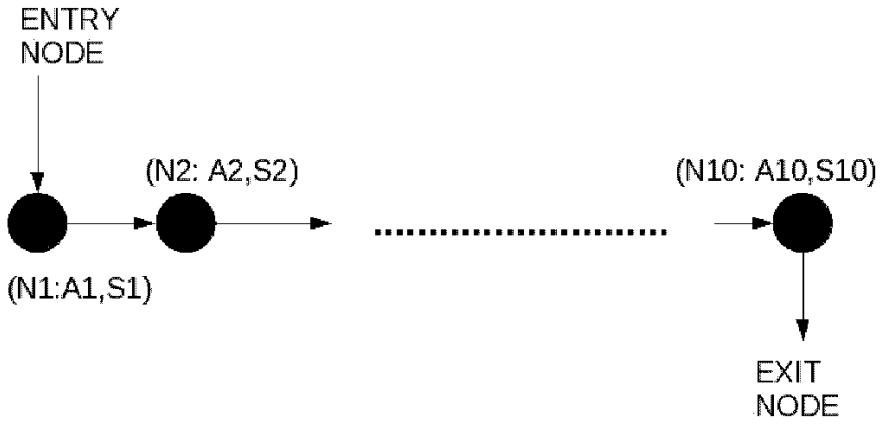


Figure 1.1 Tandem Queue Schematic

[1] Here, $\stackrel{\circ}{=}$ represents independent and identical in distribution

1.3 Construction of $\lambda(t)$

Consider the arrival process over the first period. Conditioning on A(D)=n, the arrival times on (0,D) have the same distribution as the order statistics of 'n' independent random variables with density function given by,

$$f(t) = \frac{\lambda(t)}{\Lambda(D)}, 0 \leq t \leq D. \tag{1.4}$$

The intensity function is expressed as ,

$$\lambda(t) = \begin{cases} f(t)\Lambda(D), & 0 \leq t \leq D \\ \lambda(t - kD), & kD \leq t \leq (k+1)D \;\; k = 0, 1, 2, \ldots \end{cases} \tag{1.5}$$

We construct an intensity function by choosing an appropriate probability density *f(t)* with support (0,D) and a mean number of arrivals $\Lambda(D)$ over a given period $(kD, (k+1)D), k = 0, 1, 2, \ldots$.

Though there is an ample amount of continuous densities for the choice of *f(t)*, a density that is flexible in introducing non-stationarity in the intensity function $\{\lambda(t), 0 \leq t \leq D\}$ is required here. This is because the stationary properties of the non-homogeneous Poisson Process is entirely dependent on the choice of the intensity function.

In order to serve this purpose, we employ a mixture of Beta densities defined as follows,

$$A(t) = \sum_{i=1}^{k} p_i B_i(t) \tag{1.6}$$

$$\sum_{i=1}^{k} p_i = 1, \tag{1.7}$$

where, $B_i(t)$ is Beta Density with parameters α_i and $\beta_i, t \in (0, D)$ and p_i, is a vector defining mixture probabilities. In order to verify our conjecture presented in Section 3.1, a uniform density is used for *f(t)* which will result in a stationary intensity function.

The choice of probability densities used to generate the service times are a bit more unrestricted and can be chosen from the exponential family. In our analysis, the Gamma and Uniform densities served as suitable choices.

CHAPTER 2

ARRIVAL TIME SIMULATION

The arrival process is a non-homogeneous Poisson process whose intensity function is periodic over a period (kD,(k+1)D), k=0,1,2,.... The intensity over the period is proportional to a mixture of beta densities as defined in the previous chapter. In this chapter, we outline a method employed for simulating arrivals from a non-homogeneous Poisson process by selectively thinning a homogeneous Poisson process. An unabridged description of this method is detailed in [1].

2.1 Thinning a Poisson Process

Let $\lambda_{max} \geq \max_t\{\lambda(t); 0 \leq t \leq D\}$ and let $\{N_H(t); t \geq 0\}$ be a homogeneous Poisson process with rate λ_{max}. The inter-arrival times for the above process $\{N_H(t); t \geq 0\}$ are independent and identically distributed with an exponential density having mean λ_{max}^{-1}.
We simulate $\{N_H(t); t \geq 0\}$ by generating i.i.d exponentially distributed random variables. We obtain the non-homogeneous arrival process $\{N(t); 0 \leq t \leq D\}$ by selectively sampling (thinning) the simulated path of $\{N_H(t); 0 \leq t \leq D\}$ by accepting an arrival at time $t \geq 0$ with probability $\lambda(t)/\lambda_{max}$.
For the n^{th} arrival of $\{N_H(t); 0 \leq t \leq D\}$, a random number U_n is generated from a uniform(0,1) density. The n^{th} arrival occuring at time $\{0 \leq t \leq D\}$ is accepted based on the following decision,

$$U_n \leq \lambda(t)/\lambda_{max};\ \textit{accept arrival} \tag{2.1}$$

$$U_n > \lambda(t)/\lambda_{max};\ \textit{reject arrival} \tag{2.2}$$

The accepted arrivals that are thinned out in such a manner will form the desired non-homogeneous Poisson process $\{N(t); 0 \leq t \leq D\}$ that can be employed in the tandem queue system as discussed in Chapter 1.

2.2 Simulation Results

In this section, we perform simulations that will demonstrate the thinning of a Poisson process. The period for the simulation is defined as D=1. Recall from equation (1.5) that $\lambda(t) = \Lambda(D)f(t)$, here we take *f(t)* to be a mixture of beta densities with the parameter set,

$$\mathbf{b}_\theta : \alpha = [5\ 15\ 17], \beta = [15\ 7\ 8], p = [0.3\ 0.4\ 0.3]. \tag{2.3}$$

To guarantee that $\lambda(t)$ is bounded, we assume that $\alpha_k > 1$ and $\beta_k > 1$ in each of the beta densities. Since the traffic intensity function $\lambda(t)$ also depends on $\Lambda(D)$ we choose $\Lambda(D) = 200$. We still have to choose λ_{max}:

$$\lambda_{max} \geq \max_{0\leq t\leq D} \Lambda(D)f(t) = \Lambda(D) \max_{0\leq t\leq D} f(t) \tag{2.4}$$

$$\Rightarrow \lambda_{max} \geq \Lambda(D)f^*. \tag{2.5}$$

where,

$$f^* = \max_{0\leq t\leq D} f(t).$$

It is interesting to observe that for a Beta density with $\alpha > 1$ and $\beta > 1$ $\max_{0\leq t\leq D} f_i(t)$ occurs at,

$$t_i = \frac{\alpha_i - 1}{\alpha_i + \beta_i - 2}, \forall i = 1, 2, 3 \tag{2.6}$$

$$\Rightarrow f^* \leq p_1 f(t_1) + p_2 f(t_2) + p_3 f(t_3). \tag{2.7}$$

Now, we can choose $\lambda_{max} = \Lambda(D)f^*$. Alternatively, we can plot *f(t)* using computational packages such as MATLAB or GNU Octave to choose an approximate value for f^* by eyeballing the plot. The advantage of this second approach is that it provides values smaller than the actual values for f^* and λ_{max} thereby improving the efficiency of the simulation by reducing computational complexity.

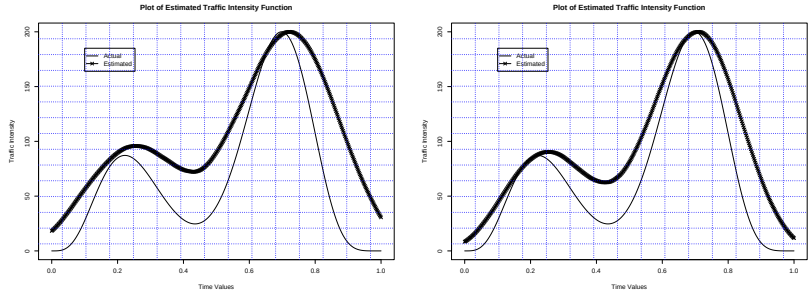
2.3 Testing the Arrival Process

Before simulating the queueing network we perform a check on simulating the arrival process. To this end, we simulate the arrival process for N_D days and estimate Λ_D and intensity function $\lambda(t)$. Here, each day is a period of length 'D'. Recall, $\Lambda_D = 200$ and the parameters of *f(t)* are given in (2.3). Table 2.1 gives the estimated values of Λ_D (denoted as $\hat{\Lambda}_D$ for different values of N_D. The Epanechnikov kernel is used to obtain the kernel

N_D	5	10	15	20	25	30	35	40	45
$\hat{\Lambda}(D)$	209.20	208.20	206.33	209.00	207.36	206.07	207.46	207.47	209.29

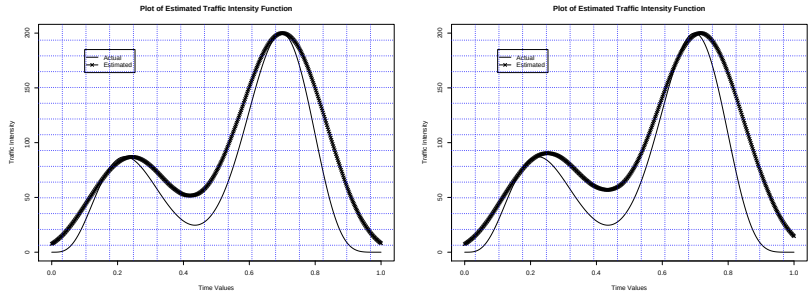
Table 2.1 Estimated values of $\Lambda(D)$ for different choices of N_D

estimate $\hat{\lambda}(t)$. The actual and estimated intensities are shown in Figure 2.1. We see that as N_D increases $\hat{\lambda}(t)$ is approaching $\lambda(t)$. However, it should be noted that both the kernel estimate and $\hat{\Lambda}(D)$ are slightly overestimated.



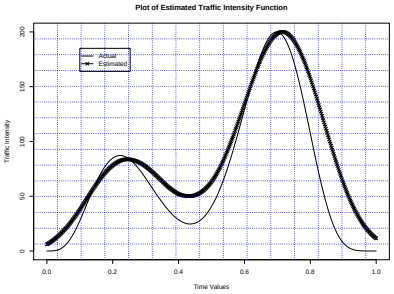
(a) N_D=5 (b) N_D=10

Figure 2.1 Kernel Density Estimation of the intensity function

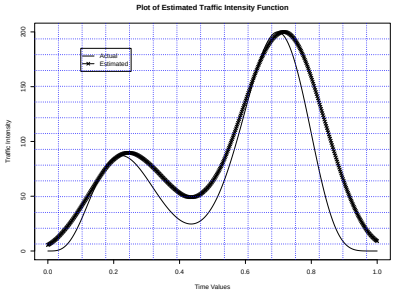


(a) N_D=15

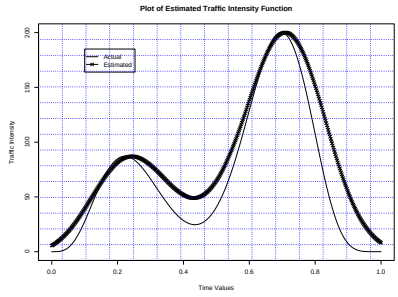
(b) N_D=20



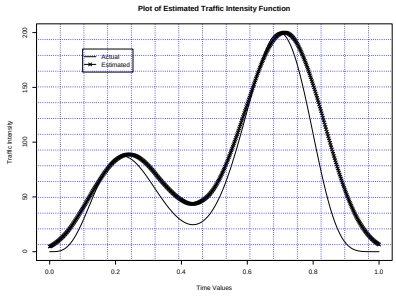
(c) N_D=25



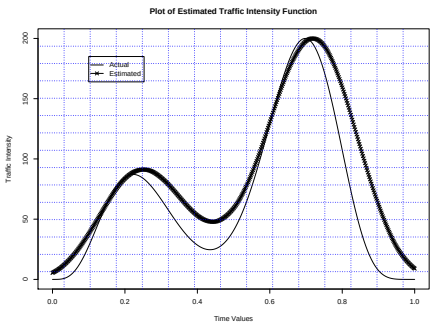
(d) N_D=30



(e) N_D=35



(f) N_D=40



(g) N_D=45

Figure 2.2 Cont'd:Kernel Estimation of a Thinned Poisson Process

CHAPTER 3

TANDEM QUEUES

We consider a tandem queueing network with 8 single server queues as discussed in the first chapter. The current chapter addresses the conjecture proposed below. In addition, Monte-Carlo simulations are performed whose results support the conjecture.

3.1 Conjecture on Tandem Queues

The queue length process at the j^{th} queue, $j = 1, 2, \ldots 8$, is asymptotically, periodically stationary. That is, there exists a periodic stationary process $\{\overline{Q_j}(t)\,; t \geq 0\}$ having period 'D' such that for all positive integers 'k', we have $\{0 \leq t_k \leq D\}$ and the number of non-negative integers $\{n_1, n_2, \ldots n_k\}$ so that,

$$\lim_{m\to\infty} P(Q_j(mD + t_1) = n_1, \ldots, Q_j(mD + t_k) = n_k) \quad = \quad P(\overline{Q_j}(t_1) = n_1, \ldots, \overline{Q_j}(t_k) = n_k).$$

For $0 \leq t \leq D$ we define

$$m_j(t) \quad = \quad E[\overline{Q_j}(t)]. \tag{3.1}$$

and,

$$m_j \quad = \quad \frac{1}{D}\int_0^D m_j(t)dt. \tag{3.2}$$

Now, if $\{\overline{Q_j}(t); t \geq 0\}$ is stationary, then the following condition holds true:

$$M_j \quad = \quad \sup_{0\leq t\leq D} |m_j(t) - m_j| = 0. \tag{3.3}$$

That is, the expected queue length as a function of time is a constant.
We conjecture that,

$$M_j \longrightarrow 0 \quad , as \quad j \to \infty \tag{3.4}$$

For the purpose of simplicity in the design and control of queueing networks, it would be useful to know whether $M_j \approx 0$ for 'j' of moderate size.

It can be observed from the simulations discussed later, in our network with 8 servers we see that M_6,M_7 and M_8 are approximately zero.

3.2 Simulation

We know, when the arrival process is stationary and traffic intensities are less than 1, then the joint queue length process $\{Q_j(t_1, t_2, \ldots t_k), 0 < t_i < D, \forall i = 1, 2, \ldots k\}$ is asymptotically stationary. In our simulation, we begin the analysis by simulating a network with an arrival stream that is stationary. We check that the asymptotic mean queue length $m_j(t)$ is constant. Next, we consider the case where the arrival process is a periodic Poisson process as discussed in Chapter 2.

We use the method proposed in [Page 615;Ross,2000] for generating the arrivals. The method is outlined in Chapter 2. At the first queue, the asymptotic mean queue length function $m_1(t)$ has a shape similar to the traffic intensity function $\lambda(t)$, that is

$$m_1(t) \approx \lambda(t).$$

A similar case is observed at queues 2,3,4 and 5 but to a lesser extent.

However, the asymptotic mean queue is nearly constant in queues 6, 7 and 8. This implies that,

$$\begin{aligned} m_j(t) &\approx m_j, 0 \leq t \leq D. \\ \Rightarrow M_j &\approx 0. \end{aligned}$$

Here, $m_j(t), m_j$ and M_j are used as defined in equations (3.1), (3.2), and (3.3). We provide simulation results corresponding to these two cases in the next section. A visual inspection of the figures[Figure 3.1,3.2] provides support for our conjecture. A statistical analysis is carried out in the next chapter which offers additional support to our conjecture.

3.3 Stationary Queue Length Process

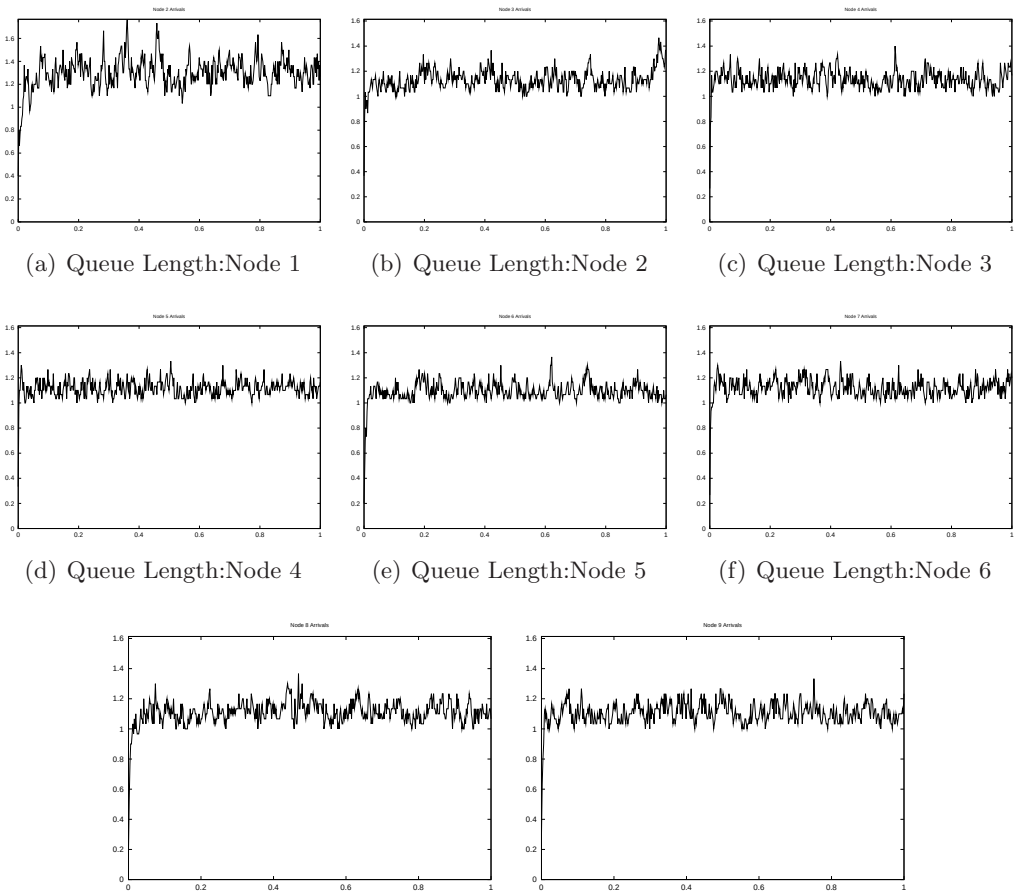
In this section, we provide results corresponding to the simulation of a stationary queue length process. The traffic intesity function is defined as,

$$\lambda(t) \quad = \quad \begin{cases} \frac{\Lambda(D)}{D}, \\ 0, \textit{otherwise}. \end{cases} \tag{3.5}$$

The service times are chosen from a uniform density with support over the interval (1,4) whose results are shown in Figure 3.1 . A gamma density with $\alpha = 2$ and $\beta = 2$ is also used to obtain service times but the results are not shown in this chapter. $\Lambda(D)$ is set to 200 and repeated for N_D= 30 days in order to fulfil asymptotic requirements. The queue length process obtained for each of the single server queues is obtained after averaging over the repititions.

3.4 Non-Stationary Queue Length Process

In this section, a non-homogeneous Poisson process $\{N(t), 0 \leq t \leq D\}$ is used as the arrival process at the first single server queue. From chapter 2, it can be noted that the intensity function employs a mixture of beta densities for *f(t)*. The queue length process at each of the single server queues is obtained using a scheme similar to the previous case.

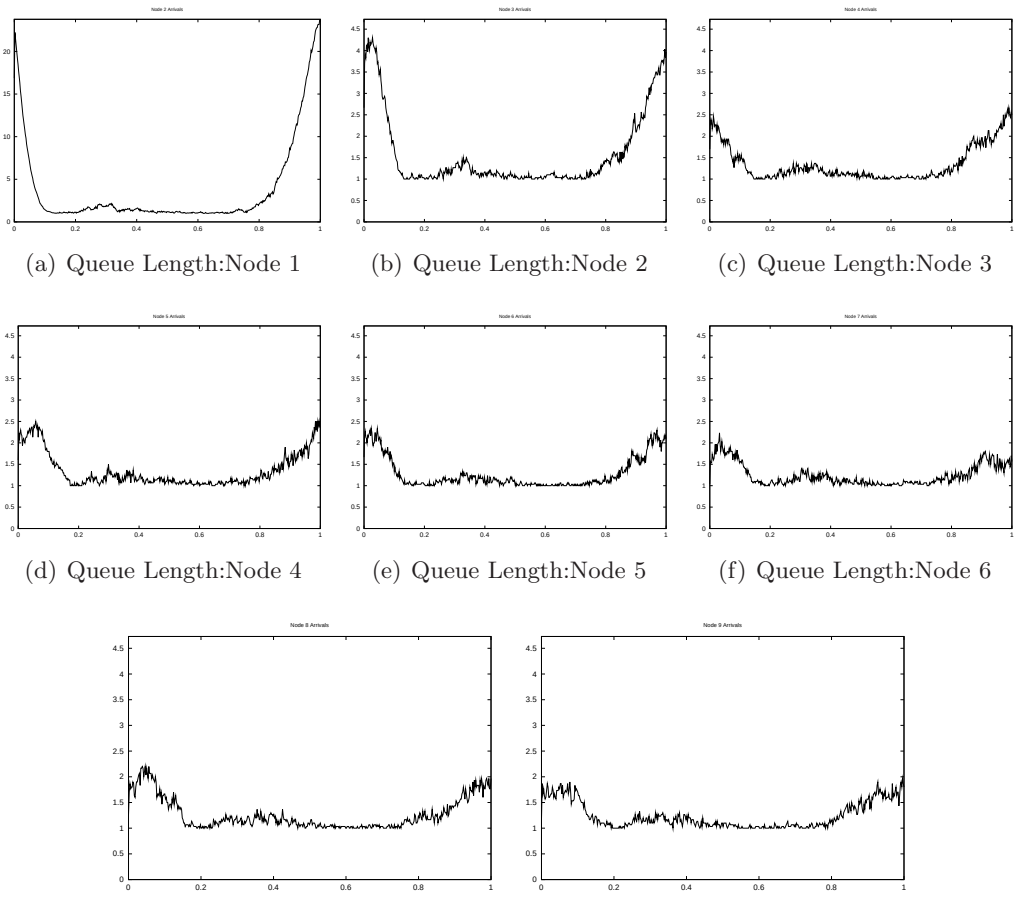


(a) Queue Length:Node 1 (b) Queue Length:Node 2 (c) Queue Length:Node 3

(d) Queue Length:Node 4 (e) Queue Length:Node 5 (f) Queue Length:Node 6

(g) Queue Length:Node 7 (h) Queue Length:Node 8

Figure 3.1 Tandem Queue Simulation-Stationary Queue Length Process



(a) Queue Length:Node 1 (b) Queue Length:Node 2 (c) Queue Length:Node 3

(d) Queue Length:Node 4 (e) Queue Length:Node 5 (f) Queue Length:Node 6

(g) Queue Length:Node 7 (h) Queue Length:Node 8

Figure 3.2 Tandem Queue Simulation-Non-Stationary Queue Length Process

CHAPTER 4

TUKEY TEST FOR STATIONARITY

The previous chapter outlined results obtained using a Monte-Carlo simulation that supported our conjecture. Here we design a test of hypothesis for our conjecture.

4.1 Design of the Test

We design a test of hypothesis for the stationarity of the asymptotic mean queue length. The basic idea involves testing for statistical similarity between sequentially isolated portions (called as 'windows') of the queue length process. We accomplish this by using a modified version of the standard Tukey Test[3].

4.1.1 Windowing

Consider the queue length at the j^{th} server $\{Q_j(t);\ 0 \leq t \leq D\}$ obtained through a non-homogeneoues Poisson Process $\{N(t);\ 0 \leq t \leq D\}$. Using vector notation, the queue length at the j^{th} server is given by $\{Q_j(t_1), Q_j(t_2), \ldots Q_j(t_{1000})\}$ where $Q_j(t_k);\ 0 \leq t_k \leq D$ is the queue length at time t_k; $\forall k = 1, 2, \ldots 1000$.

As we mentioned previously, we apportion the vector $\{Q_j(t_1), Q_j(t_2), \ldots Q_j(t_{1000})\}$ into $'\omega'$ subvectors of equal time given by $\Delta_\omega = 1000/\omega$. It should be noted that the standard Tukey test [2] involves the comparison of a certain number of factors. In our test, the $'\omega'$ subvectors corresponding to each individual segment of length Δ_ω will be treated as factors $\delta_1, \delta_2, \ldots \delta_\omega$. A data matrix is assembled such that an individual row corresponds to $\delta_m, m = 1, 2 \ldots \omega$, and an individual column corresponds to time t_n.

Thus, we define,

$$X_{in} = Q_j(\Delta_\omega(i-1) + t_n). \tag{4.1}$$

where,

$j = 1, 2 \ldots 8, i = 1, 2, \ldots \omega$ and $n = 1, 2, \ldots \Delta_\omega$

Hence, we obtain a $\omega \times \Delta_w$ matrix, X. A similar approach is taken for the stationary case by using $\lambda(t)$ as defined in (3.5).

4.2 Formal Definition of the Hypothesis Test

A Tukey test [2] assumes that the factors $\delta_1, \delta_2, \ldots \delta_\omega$ are independent . Under this assumption, we can test for the equality of means between factors. The mean queue length of window Ω for the queue length $\{Q_j(t); 0 \leq t \leq D\}$ observed at the j^{th} server is obtained as,

$$\vartheta_{\delta_\Omega} \approx \sum_{k=1}^{\Delta_\omega} \frac{X_{\Omega,k}}{\Delta_\omega}, \tag{4.2}$$

where $\Omega = 1, 2, \ldots \omega$.

$$\mathbf{H_0} : \vartheta_{\delta_1} = \vartheta_{\delta_2} = \ldots = \vartheta_{\delta_\omega} \tag{4.3}$$

$$\mathbf{H_1} : \vartheta_{\delta_1} \neq \vartheta_{\delta_2} \neq \ldots \neq \vartheta_{\delta_\omega} \tag{4.4}$$

If we reject the null hypothesis, we conclude that the factor means are different. Thus, under H_0, we can conclude that the mean queue length between isolated windows of the queue length $\{Q_j(t); 0 \leq t \leq D\}$ process observed at the j^{th} server is almost identical. Using (3.3), under the null hypothesis we cannot conclude

$$M_j \nsim 0$$

4.3 Independent Factors

The Tukey test discussed above is based on a normality assumption that each row of the assembled matrix 'X' has a normal distribution with mean $\Psi_{i.}$ and variance $\Theta_{i.}$. Also, the test assumes that the factors $\delta_1, \delta_2, \ldots \delta_\omega$ are independent of each other. However, the independence of the factors $\delta_1, \delta_2, \ldots \delta_\omega$ is violated. In order to attain independence of factors we clip portions of length ξ at the beginning and end of each window. Thus, the final data matrix is of size $[\omega, \Delta_\omega - 2\xi]$.

Also, a simple transformation is carried out to guarantee the independence of factors.

The transformation makes use of the normality assumption to attain independence of the factors. We can see that,

$$X_{i,.} - \Psi_{i,.} \sim N(0, \Theta_{i,.}) \tag{4.5}$$

$$\Rightarrow Y_{i,.} = \Theta_{i,.}^{-\frac{1}{2}}(X_{i,.} - \Psi_{i,.}) \sim N(0, I) \tag{4.6}$$

$$\Rightarrow Y_{i,.} + \Psi_{i,.} \sim N(\Psi_{i,.}, I). \tag{4.7}$$

where, $X_{i,.}$: the i^{th} row of the assembled data matrix corresponding to dependent factor $\delta_i, i = 1, 2 \ldots \omega$ of size $1 \times \Delta_\omega - 2\xi$

$Y_{i,.}$: the i^{th} row of the transformed data matrix corresponding to independent factor $\delta_i, i = 1, 2 \ldots \omega$ of size $1 \times \Delta_\omega - 2\xi$

$\Psi_{i,.}$: the i^{th} factor mean

$\Theta_{i,.}$: the estimated autocovariance matrix of size $\Delta_\omega - 2\xi \times \Delta_\omega - 2\xi$ corresponding to the i^{th} factor. Thus, a simple transformation as described in (4.7) results in a matrix in which the factors are made independent of each other.

4.4 Estimation of $\Theta_{i,.}$

In order to perform the transformation listed in (4.7) we need the autocovariance matrix,$\Theta_{i,.}$. It should be noted that each window can be treated as a time-series of length $\Delta_\omega - 2\xi$, where ξ corresponds to the length of the sample clipped at each end. Thus, the autocovariance matrix can be obtained as,

$$\hat{\theta}_i(h) = [\Delta_\omega - 2\xi]^{-1} \sum_{t=1}^{\Delta_\omega - 2\xi - |h|} [X_{i,(t+|h|)} - \vartheta_{\delta_\omega}][X_{i,t} - \vartheta_{\delta_\omega}] \tag{4.8}$$

Here, 'h' is the lag factor employed in usual autocovariance computations. Thus, the autocovariance matrix is given by,

$$\Theta_{i,.} = \begin{bmatrix} \hat{\theta}_i(0) & \hat{\theta}_i(1) & \ldots & \hat{\theta}_i(\Delta_\omega - 2\xi - 1) \\ \hat{\theta}_i(1) & \hat{\theta}_i(0) & \ldots & \hat{\theta}_i(\Delta_\omega - 2\xi - 2) \\ \vdots & \vdots & \ldots & \vdots \\ \hat{\theta}_i(\Delta_\omega - 2\xi - 1) & \hat{\theta}_i(\Delta_\omega - 2\xi - 2) & \ldots & \hat{\theta}_i(0) \end{bmatrix} \tag{4.9}$$

4.5 Matrix Computations

It can be seen from equation (4.6) that the transformation required to make the factors independent involves the square root of a covariance matrix. It should be noted that the covariance matrix must be nonnegative definite in order to obtain its square root. Most commonly used methods such as the eigenvalue , Schur and singular value decompositions rely on the nonnegative definite property of the autocovariance matrix. On the contrary, the square root of any matrix irrespective of its nonnegative definite property can be obtained using the Denman-Beavers recursive method[3][4]. The recursive method performs quite well with matrices of large dimensions and appeared to be effective in our simulation. A detailed description of the algorithm is discussed in [3].

4.6 Results of Hypothesis Testing

In this section, we carry out testing the hypothesis as stated in (4.3) and (4.4) using a standard Tukey test. We make use of the **laercio** user package in **R**.

4.6.1 Stationary Queue Length Process

The queue length process $\{Q_j(t), 0 \leq t \leq D\}$ obtained at the j^{th} server using *f(t)* to be a uniform density function is employed here. We conduct our analysis using Δ_ω=80 and ξ=10. The mean queue length for each factor is classified in to several levels using letters 'a','b',...'z'. This standard for classifying the factor levels is built within the 'laercio' package [9]. From the results shown in 4.6.1, we can see that all the factors are classified in level 'a' indicating that the mean queue length is a constant. Thus, using the definitions from chapter 3, we observe that,

$$m_j(t) \approx \Delta_\omega.a, 0 \leq t \leq D \tag{4.10}$$

$$\Rightarrow M_j \approx 0 \tag{4.11}$$

So, we reject the alternative hypothesis and conclude that the queue length process at the j^{th} server is stationary using a 95% confidence level.

Table 4.1

Tukey Test Results for a Stationary Queue Length Process

Server	δ_1	δ_2	δ_3	δ_4	δ_5	δ_6	δ_7	δ_8	δ_9	δ_{10}
1	a	a	a	a	a	a	a	a	a	a
2	a	a	a	a	a	a	a	a	a	a
3	a	a	a	a	a	a	a	a	a	a
4	a	a	a	a	a	a	a	a	a	a
5	a	a	a	a	a	a	a	a	a	a
6	a	a	a	a	a	a	a	a	a	a
7	a	a	a	a	a	a	a	a	a	a
8	a	a	a	a	a	a	a	a	a	a

4.6.2 Non-Stationary Queue Length Process

In this section we make use of non-homogeneous Poisson process involving a mixture of beta densities for *f(t)*. The queue length process obtained is segmented using the windowing scheme discussed in the previous section. It can be observed from the following table that nodes '1' through '5' expressed three different factor levels indicating that the mean queue length was not a constant. Thus, by using definitions in chapter 3 we can observe that, for some $\epsilon > 0$

$$|m_j(t) - m_s(t)| \quad > \quad \epsilon \ 0 \leq t, s \leq D, \tag{4.12}$$

$$\Rightarrow M_j \quad \neq \quad 0 \tag{4.13}$$

Therefore, we can conclude that the null hypothesis is rejected at a 95% confidence level. However, we should note that the mean queue length for servers 6,7 and 8 belong to the

same factor level 'a' for $\delta_1, \delta_2, \ldots \delta_\omega$.

Thus, it satisfies equation (4.10). Now, we partition the set of servers into ζ: {1,2,3,4,5} and φ:{6,7,8}. It can be noted that the null hypothesis is rejected for ζ whereas the null hypothesis is accepted for φ.

Hence, we can conclude that stationarity is attained as we move along the queue in a tandem queue network supporting our conjecture.

Table 4.2

Tukey Test Results for a Non-Stationary Queue Length Process

Server	δ_1	δ_2	δ_3	δ_4	δ_5	δ_6	δ_7	δ_8	δ_9	δ_{10}
1	b	c	c	c	c	c	c	c	c	a
2	a	b	b	b	b	b	b	b	b	a
3	a	c	c	c	c	c	c	c	c	b
4	a	b	b	b	b	b	b	b	b	b
5	a	b	b	b	b	b	b	b	b	b
6	a	a	a	a	a	a	a	a	a	a
7	a	a	a	a	a	a	a	a	a	a
8	a	a	a	a	a	a	a	a	a	a

4.7 Conclusion

In our work, we conjectured on the stationarity of the queue length process. Our conjecture was supported by Monte-Carlo simulations performed using Non-homogeneous Poisson processes with intensities that are non-stationary and stationary. In addition, a standard Tukey test was also performed for the stationary and non-stationary traffic intensities whose results were in favor of our conjecture. The results of the kernel estimation of the traffic intensity corresponding to a thinned Poisson process were satisfactory. However, further advances can be obtained in reducing the noticeable over-estimation observed in the kernel estimation. Also, additional research can be performed in obtaining the UMP test for stationarity.

Finally, a thorough computational analysis of the complexity and efficiency for the Monte-Carlo simulations and Matrix Computations(discussed in Chapter 4) can also be carried out to improve the significance of our analysis.

BIBLIOGRAPHY

1. Sheldon M. Ross. Introduction to Probability Models, 2000.

2. NIST/SEMATECH. E-Book of Statistical Methods: Tukey's Method.

3. Denman , Eugene D.Beavers, Alex N. *"The Matrix Sign function and computation in systems"*, Applied Mathematics and Computation **2** (1): 63-94, 1976.

4. Cheng, Sheung Hun, Hingham Nicholas J; Kenney Charles S; Laub Alan J. *"Approximating the Logarithm of a Matrix to Specified Accuracy"*, SIAM Journal on Matrix Analysis and Applications, 2001.

5. P.J Brockwell, R.A.Davis. Introduction to Time Series and Forecasting,Springer-Verlag New York, 2002.

6. V.G.Kulkarni. Modeling and Analysis of Stochastic Systems, Chapman-Hall/CRC, 1995.

7. F.P.Kelly. Reversibility and Stochastic Networks, 1979.

8. Allan Gut. An Intermediate Course in Probability, 2008.

9. Laercio Juno Da Silva. The **laercio** Package. Online Reference: *http://cran.r-project.org/web/packages/laercio/index.html*, 2008.

APPENDIX

GNU Octave Code: Mixture of Beta Densities

```
function f=compbetapdf(x,p,a,b);
%function f=compbetapdf(x,p,a,b)[Compound Beta PDF]
%x— Input vector
%p— Mixture probabilities
%a— Beta Scale parameter Vector
%b— Beta location parameter Vector
S=0;
for i=1:length(p)
S=S+p(i)*betapdf(x,a(i),b(i));
end
f=S;
```

GNU Octave Code: Thinning a Poisson Process

```
function g=poigenminmaxcrc(Ta,Tb,N,D,K,Lambda)
%************************************************************
%function g=poigenminmaxcrc(Ta,Tb,N,D,K,Lambda)
%Generate Poisson Arrival times based on lambda_min and lambda_max
%Distribution is Specified through 'D'
%Ta—Beginning point for interval
%Tb—End point for the interval
%N—Number of sample points per interval
%a—Scaling parameter of beta distribution
%b—Location parameter of beta distribution
%K—Shifting parameter for the Beta distribution
%Lambda—Number of arrivals over the interval
%f—Vector of generated arrival times
%************************************************************
f=K+D;%Generate the distribution
lambda=f.*Lambda;
%Step II—Find maximum of hazard function
```

```
lambda_min=min(lambda);
lambda_max=max(lambda);
%Step III—Generate Exponential Random Variable
E=exprnd(1/(lambda_max),1,N+1);
A(1)=E(1);
temp=A(1);
k=2;
while(k>1 & k<N+1)
   svar=0;
   for i=1:k
       svar=svar+E(i);
   end
   A(k)=svar; %Arrival  times[Several of them]
   if(A(k)≥Tb)
    A(k)=Tb—unifrnd(0.1,0.3,1,1);
   end
  k=k+1;
end
%Number of arrivals between 0 and 'T'
Na=length(A);
%Generate uniform numbers
U=rand(Na);
for i=1:Na
     if(U(i)≤(D(i)*Lambda/(lambda_max)))
        I(i)=1;
     else
        I(i)=0;
     end
end
k=1;
for i=1:Na
     if(I(i)==1)
        S(k)=i;
        k=k+1;
     end
```

```
end
for m=1:length(S)
   Ar(m)=A(S(m));
end
g=Ar;

% END CODE %
```

GNU Octave Code: Service Time Selection

```
function f=servermu(N,dchoose)
% N— Node Number
% Designed for a maximum of 10 nodes
if(dchoose==1)
X=[unifrnd(1.0,4.0,1,1) unifrnd(1.0,4.0,1,1) unifrnd(1.0,4.0,1,1) ...
unifrnd(1.0,4.0,1,1) unifrnd(1.0,4.0,1,1) unifrnd(1.0,4.0,1,1)...
unifrnd(1.0,4.0,1,1) unifrnd(1.0,4.0,1,1) unifrnd(1.0,4.0,1,1)...
unifrnd(1.0,4.0,1,1)];
elseif(dchoose==2)
X=[unifrnd(0.1,1.5,1,1) unifrnd(0.1,1.5,1,1) unifrnd(0.1,1.5,1,1)...
unifrnd(0.1,1.5,1,1) unifrnd(0.1,1.5,1,1) unifrnd(0.1,1.5,1,1)...
unifrnd(0.1,1.5,1,1) unifrnd(0.1,1.5,1,1) unifrnd(0.1,1.5,1,1)...
unifrnd(0.1,1.5,1,1)];
else
X=[gamrnd(2,2,1,1) gamrnd(2,2,1,1) gamrnd(2,2,1,1) gamrnd(2,2,1,1) ...
gamrnd(2,2,1,1) gamrnd(2,2,1,1) gamrnd(2,2,1,1) gamrnd(2,2,1,1) ...
gamrnd(2,2,1,1) gamrnd(2,2,1,1)];
end
if(N<10)
f=X(N);
elseif N==10
f=X(length(X));
else
error("Invalid Node Number");
end
```

GNU Octave Code: Tandem Queue Design

```
function [fX,fC,fD,fDn,LEN,fdX,fT,ferrval,fstep,cTime,fEtime]=ctandem10...
(lambda,p,alp,bet,P,mfac,scalefac,muo,mut,Eflag,Dflag,dchoose)
%*********************************************************************
%function [fX,fC,fD,fDn,LEN]=tandem10(lambda,p,alp,bet,P,mfac,scalefac,...
muo,mut,Eflag)
% 10-Node  Tandem Queueing Network , 8 Single Server + 2 Dummy nodes
%lambda- number of arrivals
%p-probability vector for mixture distribution[Beta]
%alp- scale parameter values
%bet- location parameter values
%P- Queue Selection probability Vector  in Ascending Order
%mfac- multiplication factor for memory effects
%scalefac-Scale factor for arrival times
%fX-Matrix with number of arrivals at each node
%fC-Matrix with arrival times at each nodes
%fD- Matrix with minimum arrival times at each node
%LEN- Returns the row length of the output matrices
%fDn- Returns the density function
%fdX- Returns the grid of the density function
%fT- Returns the simulated arrival times
%errval=Error in Queue if value is '1.0'
%muo- Mean scalar for node 2
%mut- Mean scalar for node 4
%Eflag- If set with value '1' server '2' and '9' are infinite servers
%Dglaf- If set with value '1' choose non stationary beta distribution
%else stationary
%dchoose -Chooses service time distribution
%*********************************************************************
dX=linspace(0,1,mfac*lambda);
if(Dflag==1)
Dn=compbetapdf(dX,p,alp,bet);
else
```

```
Dn=unifpdf(dX,0,1);
end
T=poigenminmaxcrc(0,1,mfac*lambda,Dn,1,lambda);
errval=0.0;
%Scaled Arrival Times
ScT=scalefac*T;
lT=length(T);
X=zeros(3*lT,10);
C=zeros(3*lT,10);
F=zeros(3*lT,10);
D=zeros(1,3*lT);
X(1,1)=1;
C(1,1)=ScT(2);
F(1,1)=1;
D(1)=nzmin(abs(C(1,:)));
dumF=sum(F(1,:));
i=1;
cTime=C(1,1);
tTime=D(1);
Etime(1)=tTime;
while(tTime>0.0 & tTime≤1000.0)
 J=X(i,1);
  if(J<lT)
    if(D(i)==C(i,1))
      X(i+1,:)=X(i,:)+[1 1 0 0 0 0 0 0 0 0];
        if(Eflag==1)
         E2=qexprndf(X(i+1,2),muo,1,1);
    else
     E2=servermu(2,dchoose);
        end
      F(i+1,:)=makeind(X(i+1,:));
      C(i+1,:)=[(ScT(J+1)−ScT(J))*F(i,1)  E2*(1−F(i,2))+(C(i,2)−D(i))*F(i,2) ...
      (C(i,3)−D(i))*F(i,3) (C(i,4)−D(i))*F(i,4) (C(i,5)−D(i))*F(i,5)...
      (C(i,6)−D(i))*F(i,6) (C(i,7)−D(i))*F(i,7) (C(i,8)−D(i))*F(i,8) ...
      (C(i,9)−D(i))*F(i,9) (C(i,10)−D(i))*F(i,10)];
      C(i+1,:)=abs(C(i+1,:));
     elseif (D(i)==C(i,2))
```

```
        X(i+1,:)=X(i,:)+[0 -1 1 0 0 0 0 0 0 0];
         if(Eflag==1)
          E2=qexprndf(X(i+1,2),muo,1,1);
    else
     E2=servermu(2,dchoose);
    end
     S3=servermu(3,dchoose);
     F(i+1,:)=makeind(X(i+1,:));
     C(i+1,:)=[(C(i,1)-D(i))*F(i,1) E2*F(i+1,2) ...
     S3*(1-F(i,3))+(C(i,3)-D(i))*F(i+1,3)  (C(i,4)-D(i))*F(i+1,4) ...
     (C(i,5)-D(i))*F(i+1,5) (C(i,6)-D(i))*F(i+1,6) (C(i,7)-D(i))*F(i+1,7) ...
     (C(i,8)-D(i))*F(i+1,8) (C(i,9)-D(i))*F(i+1,9) (C(i,10)-D(i))*F(i+1,10)];
      C(i+1,:)=abs(C(i+1,:));
     elseif (D(i)==C(i,3))
        X(i+1,:)=X(i,:)+[0 0 (-1) 1 0 0 0 0 0 0];
        F(i+1,:)=makeind(X(i+1,:));
        S3=servermu(3,dchoose);
        S4=servermu(4,dchoose);
        C(i+1,:)=[(C(i,1)-D(i))*F(i,1) (C(i,2)-D(i))*F(i+1,2)  S3*F(i+1,3) ...
        S4*(1-F(i,4))+(C(i,4)-D(i))*F(i,4) (C(i,5)-D(i))*F(i+1,5) ...
    (C(i,6)-D(i))*F(i+1,6) (C(i,7)-D(i))*F(i+1,7)  (C(i,8)-D(i))*F(i+1,8)...
    (C(i,9)-D(i))*F(i+1,9) (C(i,10)-D(i))*F(i+1,10)];
    C(i+1,:)=abs(C(i+1,:));
    elseif (D(i)==C(i,4))
X(i+1,:)=X(i,:)+[0 0 0 -1 1 0 0 0 0 0];
F(i+1,:)=makeind(X(i+1,:));
S4=servermu(4,dchoose);
S5=servermu(5,dchoose);
C(i+1,:)=[(C(i,1)-D(i))*F(i,1) (C(i,2)-D(i))*F(i+1,2) (C(i,3)-D(i))*F(i+1,3) ...
S4*F(i+1,4) S5*(1-F(i,5))+(C(i,5)-D(i))*F(i,5) (C(i,6)-D(i))*F(i+1,6) ...
(C(i,7)-D(i))*F(i+1,7) (C(i,8)-D(i))*F(i+1,8) (C(i,9)-D(i))*F(i+1,9) ...
(C(i,10)-D(i))*F(i+1,10)];
C(i+1,:)=abs(C(i+1,:));
    elseif (D(i)==C(i,5))
X(i+1,:)=X(i,:)+[0 0 0 0  -1 1  0 0 0 0];
F(i+1,:)=makeind(X(i+1,:));
```

```
 S5=servermu(5,dchoose);
 S6=servermu(6,dchoose);
C(i+1,:)=[(C(i,1)-D(i))*F(i,1) (C(i,2)-D(i))*F(i+1,2)  (C(i,3)-D(i))*F(i+1,3) ...
 C(i,4)-D(i))*F(i+1,4) S5*F(i+1,5)  S6*(1-F(i,6))+(C(i,6)-D(i))*F(i+1,6) ...
 (C(i,7)-D(i))*F(i+1,7) (C(i,8)-D(i))*F(i+1,8) (C(i,9)-D(i))*F(i+1,9) ...
 (C(i,10)-D(i))*F(i+1,10)];
 C(i+1,:)=abs(C(i+1,:));
     elseif (D(i)==C(i,6))
 X(i+1,:)=X(i,:)+[0 0 0 0 0 -1 1 0 0 0];
 F(i+1,:)=makeind(X(i+1,:));
 S7=servermu(7,dchoose);
 S6=servermu(6,dchoose);
 C(i+1,:)=[(C(i,1)-D(i))*F(i,1) (C(i,2)-D(i))*F(i+1,2)  (C(i,3)-D(i))*F(i+1,3) ...
 (C(i,4)-D(i))*F(i+1,4) (C(i,5)-D(i))*F(i+1,5) S6*F(i+1,6)  ...
 S7*(1-F(i,7))+(C(i,7)-D(i))*F(i,7)  (C(i,8)-D(i))*F(i+1,8) ...
 (C(i,9)-D(i))*F(i+1,9) (C(i,10)-D(i))*F(i+1,10)];
 C(i+1,:)=abs(C(i+1,:));
    elseif (D(i)==C(i,7))
 X(i+1,:)=X(i,:)+[0 0 0 0 0 0 -1 1 0 0] ;
 F(i+1,:)=makeind(X(i+1,:));
 S7=servermu(7,dchoose);
 S8=servermu(8,dchoose);
 C(i+1,:)=[(C(i,1)-D(i))*F(i,1) (C(i,2)-D(i))*F(i+1,2)  (C(i,3)-D(i))*F(i+1,3) ...
 C(i,4)-D(i))*F(i+1,4) (C(i,5)-D(i))*F(i+1,5) (C(i,6)-D(i))*F(i+1,6) ...
 S7*F(i+1,7) S8*(1-F(i,8))+(C(i,8)-D(i))*F(i,8)  (C(i,9)-D(i))*F(i+1,9) ...
 (C(i,10)-D(i))*F(i+1,10)];
 C(i+1,:)=abs(C(i+1,:));
    elseif (D(i)==C(i,8))
 X(i+1,:)=X(i,:)+[0 0 0 0 0 0 0 -1 1 0];
 F(i+1,:)=makeind(X(i+1,:));
    if(Eflag==1)
       E9=qexprndf(X(i+1,9),mut,1,1);
    else
       E9=servermu(9,dchoose);
    end
        S8=servermu(8,dchoose);
        C(i+1,:)=[(C(i,1)-D(i))*F(i,1) (C(i,2)-D(i))*F(i+1,2) ...
         (C(i,3)-D(i))*F(i+1,3)  (C(i,4)-D(i))*F(i+1,4) (C(i,5)-D(i))*F(i+1,5) ...
```

```
        (C(i,6)-D(i))*F(i+1,6)  (C(i,7)-D(i))*F(i+1,7) S8*F(i+1,8)  ...
         E9*(1-F(i,9))+(C(i,9)-D(i))*F(i,9)   (C(i,10)-D(i))*F(i+1,10)];
         C(i+1,:)=abs(C(i+1,:));
    elseif (D(i)==C(i,9))
           X(i+1,:)=X(i,:)+[0 0 0 0 0 0 0 0 -1 1];
       F(i+1,:)=makeind(X(i+1,:));
       S10=servermu(1001,dchoose);
            if(Eflag==1)
             E9=qexprndf(X(i+1,9),mut,1,1);
        else
         E9=servermu(9,dchoose);
        end
       C(i+1,:)=[(C(i,1)-D(i))*F(i,1) (C(i,2)-D(i))*F(i+1,2) ...
       (C(i,3)-D(i))*F(i+1,3)  (C(i,4)-D(i))*F(i+1,4) (C(i,5)-D(i))*F(i+1,5) ...
       (C(i,6)-D(i))*F(i+1,6)  (C(i,7)-D(i))*F(i+1,7)  (C(i,8)-D(i))*F(i+1,8) ...
       E9*F(i+1,9)  S10*(1-F(i,10))+(C(i,10)-D(i))*F(i,10)];
       C(i+1,:)=abs(C(i+1,:));
   elseif (D(i)==C(i,10))
           X(i+1,:)=X(i,:)+[0 0 0 0 0 0 0 0 0 -1];
           F(i+1,:)=makeind(X(i+1,:));
       S10=servermu(1001,dchoose);
           E9=qexprndf(X(i+1,9),mut,1,1);
       C(i+1,:)=[(C(i,1)-D(i))*F(i,1) (C(i,2)-D(i))*F(i+1,2)...
      (C(i,3)-D(i))*F(i+1,3) (C(i,4)-D(i))*F(i+1,4)...
       (C(i,5)-D(i))*F(i+1,5) (C(i,6)-D(i))*F(i+1,6) ...
       (C(i,7)-D(i))*F(i+1,7)  (C(i,8)-D(i))*F(i+1,8) ...
       (C(i,9)-D(i))*F(i+1,9) S10*F(i+1,10)];
       C(i+1,:)=abs(C(i+1,:));
   else
       errval=1.0;
      %error("Facing Problems with Queue");
   endif
else
    break;

endif
     if(sum(C(i+1,:))>0)
     D(i+1)=nzmin(abs(C(i+1,:)));
```

```
      endif
      dumF=sum(F(i+1,:));
      tTime=tTime+(D(i+1));
      ETime(i)=tTime;
      curstep=i;
      i=i+1;
end
fX=X;
fC=C;
fD=D;
[LEN,LENy]=size(X);
fDn=Dn;
fdX=dX;
fT=T;
ferrval=errval;
fstep=curstep;
fEtime=ETime;
```

GNU Octave Code: Tandem Queue Design Extras

The code snippets corresponding to 'qexprndf' is excluded as it was never implemented in our simulations. It is a slightly modified version of the 'exprndf' suited to work with our simulation. Also, the function 'ctandem10ovr' employed in the simulations is a modification of the 'ctandem10' function. The latter is equipped with alterations such that arrivals from the previous simulation step can be carried over to new step. Because of its similarity to 'ctandem10', it is not included in the Appendix.

GNU Octave Code: Tandem Simulation

```
%Generate Arrival Times
clear all;
tic();
lambda=300;
d=20;
p=[0.3 0.4 0.3];
a=[5 15 17];
b=[15 7 8];
```

```
mfactor=4;
nfactor=16;
%Initialization
M2=zeros(d,nfactor*lambda);
M3=zeros(d,nfactor*lambda);
M4=zeros(d,nfactor*lambda);
M5=zeros(d,nfactor*lambda);
M6=zeros(d,nfactor*lambda);
M7=zeros(d,nfactor*lambda);
M8=zeros(d,nfactor*lambda);
M9=zeros(d,nfactor*lambda);
M10=zeros(d,nfactor*lambda);
TD=zeros(d,nfactor*lambda);
Grid=linspace(0,1000,1001);
N=[2 3 4 5 6 7 8 9 10];
% Tandem Simulation Loop
for i=1:d
  if(i==1)
    [tX,tC,tD,tDn,LEN(i),G,Gt,aerrval,cstep,cTime,aevTime]=...
    ctandem10(lambda,p,a,b,Psel,mfactor,1000,0.5,0.5,0,0,1);
    timelength(i)=length(aevTime');
    [M2(i,:),M3(i,:),M4(i,:),M5(i,:),M6(i,:),M7(i,:),...
    M8(i,:),M9(i,:),M10(i,:),TD(i,:)]
    = makeMmatrixsel(tX,aevTime',LEN(i),nfactor,lambda,N);
  else
[tX,tC,tD,tDn,LEN(i),G,Gt,berrval,cstep,cTime,aevTime]=...
ctandem10ovr(tX(cstep+1,:),tC(cstep+1,:),tD(cstep+1),...
lambda,p,a,b,Psel,mfactor,1000,0.5,0.5,0,0,1);
    timelength(i)=length(aevTime');
   [M2(i,:),M3(i,:),M4(i,:),M5(i,:),M6(i,:),M7(i,:),...
   M8(i,:),M9(i,:),M10(i,:),TD(i,:)]
   =makeMmatrixsel(tX,aevTime',LEN(i),nfactor,lambda,N);
   end
end
% timeaverage2 is used to compute average at any given time increment.
TD=TD./1000;
[N2,Grid]=timeaverage2(M2,TD,1000,timelength);
n2=sum(N2)/d;
```

```
[N3,Grid]=timeaverage2(M3,TD,1000,timelength);
n3=sum(N3)/d;
[N4,Grid]=timeaverage2(M4,TD,1000,timelength);
n4=sum(N4)/d;
[N5,Grid]=timeaverage2(M5,TD,1000,timelength);
n5=sum(N5)/d;
[N6,Grid]=timeaverage2(M6,TD,1000,timelength);
n6=sum(N6)/d;
[N7,Grid]=timeaverage2(M7,TD,1000,timelength);
n7=sum(N7)/d;
[N8,Grid]=timeaverage2(M8,TD,1000,timelength);
n8=sum(N8)/d;
[N9,Grid]=timeaverage2(M9,TD,1000,timelength);
n9=sum(N9)/d;
[N10,Grid]=timeaverage2(M10,TD,1000,timelength);
n10=sum(N10)/d;
G=Grid;
[Fid,MSG]=fopen("/media/sda7/PoissonRes/A300/d1node.csv",'w','native');
fprintf(Fid,"%s,%s,%s,%s,%s,%s,%s,%s,%s \n","n2","n3","n4","n5","n6","n7", ...
"n8","n9","Grid");
for i=1:length(n2)
fprintf(Fid,"%f,%f,%f,%f,%f,%f,%f,%f,%f \n",n2(i),n3(i),n4(i),n5(i),n6(i),...
n7(i),n8(i),n9(i),Grid(i));
end
elapsed_time=toc
# END CODE #
```

Denman-Beavers Algorithm

```
denman.beavers<-function(mat,maxit=50){
stopifnot(nrow(mat)==ncol(mat))
niter<-0
y<-mat
z<-diag(rep(1,nrow(mat)))
for (niter in 1:maxit) {
y.temp <-0.5*(y+solve(z))
z<-0.5*(z+solve(y))
y<-y.temp
```

```
}
return(list(sqrt=y,sqrt.inv=z))
}
### END SCRIPT ###
```

The code for Denman-Beavers is written in R. The script is contributed to the community under the GNU License. The scripts involved in Kernel estimation, Plotting and miscellaneous work are not included in this appendix in an attempt to keep the document concise.