Time Dependent Queuing Systems

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

Allen E. Flick

Certificate of Approval:

Bertram Zinner Associate Professor Mathematics & Statistics	Ming Liao, Chair Professor Mathematics & Statistics
Amnon J. Meir Professor Mathematics & Statistics	Geraldo S. DeSouza Professor Mathematics & Statistics

George T. Flowers
Graduate School Dean

Time Dependent Queuing Systems

Allen E. Flick

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Science

Auburn, Alabama
December 19, 2008

Time Dependent Queuing Systems

Allen E. Flick

_____

Signature of Author

_____

Date of Graduation

Vita

Allen E. Flick was born to William and Susan Flick on March 19th, 1981 in Cleveland, Ohio. Shortly thereafter, his family moved to Birmingham, Alabama, where he graduated from Vestavia Hills High School in 1999. After working for a while, he decided to try college and took a few classes at the University of Alabama at Birmingham. He then opted to go to school full time and moved to Auburn, Alabama, where he graduated Summa Cum Laude from Auburn University with a Bachelor of Science in Applied Mathematics in August, 2006. He began the Graduate program in Mathematics at Auburn University two weeks later.

Thesis Abstract

Time Dependent Queuing Systems

Allen E. Flick

Master of Science, December 19, 2008
(B.S., Auburn University, 2006)

69 Typed Pages

Directed by Ming Liao

Using elementary probability theory, we establish limiting probabilities for the queue length of a queuing system whose arrivals follow a nonhomogeneous Poisson process and are served by a single server whose services also follow a nonhomogeneous Poisson process. We uniformly accelerate the process and conclude, under a special stability condition, that the queue length distribution is the same as a queue with constant rates. Extensions are provided for queues with multiple homogeneous servers and those with a finite capacity. Also included is a simulation of such a queuing system using the data from an Auburn University web server to model the arrival process.

Acknowledgments

The author would like to first thank his adviser, Professor Liao, for his unlimited time and patience this last year. He would also like to thank the rest of his committee for their advice and support. He thanks all his professors at Auburn University, who have opened his eyes to the amazing world of mathematics. And last, but not least, the author would like to thank his family and friends for their unending support.

Style manual or journal used Journal of Approximation Theory (together with the style known as "aums"). Bibliograpy follows van Leunen's *A Handbook for Scholars.*

Computer software used The document preparation package TEX (specifically LATEX) together with the departmental style-file `aums.sty`.

Table of Contents

List of Figures

Chapter 1

Introduction

This purpose of this thesis is to provide an intuitive probabilistic approach to time dependent queuing systems. It is inspired by William A. Massey's paper, "Asymptotic Analysis of the Time Dependent M/M/1 Queue"[3]. In his work, Massey takes an analytic approach to such processes; which, while very precise and rigorous, is difficult for someone not intimately familiar with higher mathematics to understand. But the main result is the same: if we ε-accelerate the process, that is make the arrival and service rates arbitrarily large, then the time dependent $M(t)/M(t)/1$ queue acts just like a constant rate $M/M/1$ queue as time goes to infinity, although with a different requirement for stability. Instead of needing a larger service rate than arrival rate, the stability requirement for any time t is $\rho^*(t) < 1$. This means, as we will discuss in Chapter 3, that the server, on average, has time to catch up at time t. So we don't necessarily always need the service rate larger than the arrival rate, we just need the amount of possible service before time t to be able to handle the possible arrivals before time t.

Since this paper was written for a broader audience, the next chapter is a summary of some basic ideas in probability and queuing theory. It starts with a discussion of some of the main ideas in probability, then moves on to discuss Poisson processes and constant rate queuing systems. In Chapter 3, we present our main results about time-varying queuing systems, Theorems 3 and 4. They show that the time dependent queue is very similar to it's constant rate counterpart. It will begin with a definition of an $M(t)/M(t)/1$ queue and a description of the very special quantity, ρ^*, before moving on to the main proofs.

Chapter 4 includes two extensions of the main result: time dependent queuing systems with multiple servers and systems with a finite capacity. The last chapter is a simulation of a time dependent queuing system, a model of an Auburn University web server.

Chapter 2

Preliminary Information

This chapter is a collection of preliminary information necessary for the understanding of the proofs in the later chapters. It is provided so that those with only very basic knowledge of probability and stochastic processes can understand this paper. The bulk of this chapter can be found in the works of Billingsley [1], Ross [4], and Gross and Harris [2].

2.1 Basic Probability Theory

Some basic notation and definitions:

$P[A]$ is the *probability* that some event A will occur;

$$0 \leq P[A] \leq 1 \text{ for any event } A.$$

A^c is the event that is everything in the sample space except for whatever is in A;

$$P[A^c] = 1 - P[A].$$

$P[A \mid B]$ is the probability that A will occur *given* B has occurred.

The Law of Total Probability: For any event A in some sample space Ω,

$$P[A] = \sum_n P[A \mid B_n]P[B_n],$$

where $\{B_n\}$ is a countable partition of Ω.

$E[X]$ is the *expected value* of a random variable X; it is the average value of X.

Convergence in Distribution [1]: A sequence of random variables X_n is said to converge *in distribution* to a random variable X if

$$P[X_n > x] \to P[X > x] \text{ as } n \to \infty \text{ , whenever } P[X = x] = 0.$$

We say $X_n \to X$, in distribution.

Almost Sure Convergence [1]: A sequence of random variables X_n is said to converge *almost surely* to a random variable X if

$$P[X_n \to X] = 1.$$

The Strong Law of Large Numbers (SLLN)[1]: If X_n is an independently and identically distributed (iid) sequence of random variables, each with a finite mean, then

$$\frac{\sum_{i=1}^{n} X_i}{n} \to E[X_1] \text{ , almost surely, as } n \to \infty.$$

Poisson Distribution [1]: If X is a Poisson distributed random variable with mean $\lambda > 0$, then

$$P[X = k] = e^{-\lambda}\frac{\lambda^k}{k!} \text{ for } k = 0, 1, 2... \text{ , and } E[X] = \lambda.$$

Exponential Distribution [4]: If X is an exponentially distributed random variable with rate $\lambda > 0$, then

$$P[X > x] = \int_x^{\infty} \lambda e^{-\lambda t}\, dt = e^{-\lambda x} \text{ for } x \geq 0 \text{ , and } E[X] = \frac{1}{\lambda}.$$

Lack of Memory Property [4]: X is an exponentially distributed random variable if and only if it does not assume negative values and

$$P[X > t + s \mid X > t] = P[X > s].$$

2.2 Poisson Processes

A *stochastic process* is a collection of random variables indexed by time [4]. With a stochastic process X, for every time index t, $X(t)$ is a random variable that represents the state of the process at time t.

A stochastic process $N(t)$ is a *counting process* [4] if it represents the total number of events that have occurred up to time t, and therefore must satisfy

(i) $N(t) \geq 0$

(ii) $N(t)$ is integer valued

(iii) If $s < t$, then $N(s) \leq N(t)$

(iv) For $s < t$, $N(t) - N(s)$ is the number of events that have occurred in the interval $(s, t]$.

A counting process is said to have *independent increments* if the numbers of events that occur in disjoint time intervals are independent [4].

A counting process is said to have *stationary increments* if the distribution of the number of events that occur in any interval of time depends only on the length of the time interval [4].

A counting process $N(t)$ is said to be a *Poisson Process* [4] with rate $\lambda > 0$ if

(i) $N(0) = 0$

(ii) The process has stationary and independent increments

(iii) The number of events in any interval of length t is Poisson distributed with mean λt. So for all $s, t \geq 0$,

$$P[N(t+s) - N(s) = n] = e^{-\lambda t}\frac{(\lambda t)^n}{n!} \text{ for } n = 0, 1, \ldots$$

The *interarrival times*, or the times between successive arrivals, of a Poisson process $N(t)$ with rate λ, are iid exponential random variables of rate λ [4].

The counting process $N(t)$ is a *nonhomogeneous Poisson Process* [4] with rate function $\lambda(t)$ if

(i) $N(0) = 0$

(ii) The process has independent increments

(iii) For $s < t$, $N(t) - N(s)$ is Poisson distributed of mean $m(t) - m(s)$, where $m(t)$ is the process's mean function

$$m(t) = \int_0^t \lambda(s)\, ds.$$

2.3 Basic Queueing Systems

An $M/M/1$ queue is a system where arrivals, considered customers, arrive according to a homogeneous Poisson process and their required service times are iid exponentially distributed random variables, therefore the number of completed services also follows a Poisson process if the server is always busy [4]. If $X(t)$ is an $M/M/1$ queue with arrival

rate λ and service rate μ, then we call $\rho := \lambda/\mu$ the traffic intensity. We call $X(t)$ stable if $\rho < 1$, since then as $t \to \infty$ we have

$$P[X(t) = n] \to p_n = \rho^n(1 - \rho) \text{ for } n = 1, 2, \ldots$$

If $\rho > 1$, then the queue length tends to infinity as $t \to \infty$ [2]. This probability is also equal to the long run fraction of time that X is in state n, or that the queue is n customers long, almost surely. The set of all these probabilities for $n = 0, 1, \ldots$ is called the stationary distribution of X [2].

Note that since $p_0 > 0$, then the queue length will be zero at some time as $t \to \infty$, almost surely. So if τ is the first time s that $X(s) = 0$, then $P[\tau < t] \to 1$ as $t \to \infty$.

An $M/M/k$ queue is similar to an $M/M/1$ queue, but with k homogeneous servers. If $X(t)$ is an $M/M/k$ queue with arrival rate λ and k servers each with service rate μ, then we call $\rho := \lambda/(k\mu)$ our traffic intensity. Again we call $X(t)$ stable if $\rho < 1$, since then as $t \to \infty$ we have, from [2],

$$P[X(t) = n] \to p_n = \begin{cases} \dfrac{\lambda^n}{n!\mu^n} p_0 & \text{for } 1 \le n \le k \\ \dfrac{\lambda^n}{k^{n-k} k!\mu^n} p_0 & \text{for } n \ge k \end{cases}$$

$$\text{where } p_0 = \left[\frac{1}{k!} \left(\frac{\lambda}{\mu} \right)^k \left(\frac{k\mu}{k\mu - \lambda} \right) + \sum_{n=0}^{k-1} \frac{1}{n!} \left(\frac{\lambda}{\mu} \right)^n \right]^{-1}.$$

An $M/M/k/N$ queue is an $M/M/k$ queue with a finite capacity N. Let $X(t)$ be an $M/M/k/N$ queue with arrival rate λ, k servers each with service rate μ, and capacity N. If $X(t) \ge N$, then the server is full and can't accept any more arrivals, so the service rate

at that point is zero. Since we have a finite number of states, then there is a steady state distribution regardless of the arrival and service rates. So as $t \to \infty$ we have

$$P[X(t) = n] \to p_n = \begin{cases} \dfrac{\lambda^n}{n!\mu^n} p_0 & \text{for } 0 \le n < k \\ \dfrac{\lambda^n}{k^{n-k} k! \mu^n} p_0 & \text{for } k \le n \le N \end{cases}$$

$$\text{where } p_0 = \begin{cases} \left[\dfrac{1}{k!} \left(\dfrac{\lambda}{\mu} \right)^k \dfrac{1 - (\lambda/k\mu)^{N-k+1}}{1 - \lambda/k\mu} + \displaystyle\sum_{n=0}^{k-1} \frac{1}{n!} \left(\frac{\lambda}{\mu} \right)^n \right]^{-1} & \text{for } \dfrac{\lambda}{k\mu} \neq 1 \\ \left[\dfrac{1}{k!} \left(\dfrac{\lambda}{\mu} \right)^k (N - k + 1) + \displaystyle\sum_{n=0}^{k-1} \frac{1}{n!} \left(\frac{\lambda}{\mu} \right)^n \right]^{-1} & \text{for } \dfrac{\lambda}{k\mu} = 1 \end{cases},$$

from [2].

Chapter 3

$M(t)/M(t)/1$ Queues

3.1 Definition and ρ^*

Let $X(t)$ be the queue length of an $M(t)/M(t)/1$ Queue with initial state $X(t_0) = X_0$. Arrivals follow a non-homogeneous poisson process of rate $\lambda(t)$, and requests by each arrival are i.i.d. exponential random variables of mean 1. The single server works at a rate of $\mu(t)$, so $\int_a^b \mu(t)\, dt$ is the amount of work the server can do from time a to time b, or the expected number of services the server can handle in (a, b). The traffic intensity is $\rho(t) = \lambda(t)/\mu(t)$. And define

$$\rho^*(t) := \rho^*(t; \lambda, \mu) = \sup_{t_* \in (t_0, t)} \frac{\int_{t_*}^t \lambda(s)\, ds}{\int_{t_*}^t \mu(s)\, ds}.$$

ρ^* represents whether the server, on average, can handle the arrivals or not. If $\rho^*(t) < 1$, then for any time $s < t$ the expected number of services the server can handle is more than the expected number of arrivals on the interval $(s, t]$. So the server is always able to catch up, even if the arrival rate is at some times much higher than the service rate. If $\rho^*(t) > 1$, at some point the arrivals are so frequent that the server can't catch up by time t. The case where $\rho^*(t) = 1$ is beyond the scope of this paper, but is covered thoroughly by Massey [3].

Throughout this paper, we use the concept of uniform acceleration, as did Massey [3]. The idea of this acceleration, which we call ε-accelerating a process, is to divide both the arrival rate and the service rate by some small $\varepsilon > 0$. Then as $\varepsilon \to 0$, the number of arrivals and completed services in any time interval is arbitrarily large. This has the same effect on a constant $M/M/1$ queue as letting time approach infinity. In our case, it allows us to

consider the stability at time t, since we can see how the system reacts with an arbitrarily large number of arrivals and services. If $X(t)$ is an $M(t)/M(t)/1$ Queue with arrival rate $\lambda(t)$ and service rate $\mu(t)$, then the ε-accelerated process $X^{\varepsilon}(t)$ is an $M(t)/M(t)/1$ Queue with arrival rate $\lambda(t)/\varepsilon$ and service rate $\mu(t)/\varepsilon$. Since this doesn't change $\rho^*(t)$, we can use $\rho^*(t)$ make judgements about the system's stability.

3.2 The Principle of Stochastic Dominance

For $M(t)/M(t)/1$ Queues $X(t)$ and $Y(t)$ with arrival rates $\lambda_X(t)$ and $\lambda_Y(t)$ and service rates $\mu_X(t)$ and $\mu_Y(t)$, if $\lambda_X(t) \leq \lambda_Y(t)$ and $\mu_X(t) \geq \mu_Y(t)$, then

$$P[X^{\varepsilon}(t) \geq n] \leq P[Y^{\varepsilon}(t) \geq n].$$

A rigorous proof of this fact by a purely analytic method is included in Section 10 of Massey's paper [3]. Here, we provide an informal probabilistic argument.

Assume $\lambda_X(t) \leq \lambda_Y(t)$ and $\mu_X(t) = \mu_Y(t)$, then we can consider the arrival process of $Y(t)$ to have rate function $\lambda_Y(t) = \lambda_X(t) + a(t)$, where $a(t) = \lambda_Y(t) - \lambda_X(t)$. We are essentially adding an arrival stream to $X(t)$ with a mean number of arrivals $\int_{t_0}^{t} a(s)\, ds$; while the server is working at the same rate, it has more requests to handle. We may allow the server at $Y(t)$ to handle arrivals to $X(t)$ in the queue before additional arrivals due to $a(t)$. This will not change the queue length, so the queue whose length is represented by $Y(t)$ is at least as congested at any time t as the queue whose length is represented by $X(t)$. Thus $P[X^{\varepsilon}(t) \geq n] \leq P[Y^{\varepsilon}(t) \geq n]$.

Likewise, if $\mu_X(t) \geq \mu_Y(t)$ and $\lambda_X(t) = \lambda_Y(t)$, we can consider the rate of the process that represents the possible service of $X(t)$ to be $\mu_X(t) = \mu_Y(t) + b(t)$, where

$b(t) = \lambda_X(t) - \lambda_Y(t)$. Therefore the server whose queue length is represented by $X(t)$ can work at at faster rate. It can handle a mean number of requests $\int_{t_0}^{t} b(s)\,ds$ more than the queue whose length is represented by $Y(t)$. So the server in $X(t)$ is at most as congested as the server in $Y(t)$ and we have $P[X^\varepsilon(t) \geq n] \leq P[Y^\varepsilon(t) \geq n]$.

3.3 Queue Length

In the following proofs, we will assume the functions are left-continuous. This is not as much a requirement for correctness as it is an ease of notation. Without this assumption, the proofs still hold; we would simply need to consider the left-hand limit of the function or process wherever we consider the function or process at some time.

Theorem 1 *Fix $t > h > 0$. If λ and μ are left-continuous piecewise-constant positive real functions of time, with $\lambda(s) < \mu(s)$ for all $s \in [t_0, t]$, then if we consider the uniformly accelerated $M(t)/M(t)/1$ Queue $X^\varepsilon(t)$ with initial state $X^\varepsilon(t_0) = X_0$, arrival rate $\lambda(t)/\varepsilon$, and service rate $\mu(t)/\varepsilon$, we have*

$$P[X^\varepsilon(t) = n] \to (\rho(t))^n(1 - \rho(t)) \text{ as } \varepsilon \to 0$$

$$\text{and} \quad P[\Omega_\varepsilon] \to 1 \text{ as } \varepsilon \to 0,$$

where Ω_ε is the event that $X^\varepsilon(s) = 0$ for some $s \in (t - h, t)$, $h > 0$.

There exists a partition Δ of $[t_0, t]$ with $t_0 < t_1 < \cdots < t_N = t$ such that $\lambda = \lambda_i$ and $\mu = \mu_i$ for constant λ_i and μ_i on the subinterval $I_i = (t_{i-1}, t_i]$ for $1 \leq i \leq N$. On each subinterval I_i, $X = X_i$, an $M/M/1$ queue with constant arrival rate λ_i and service rate μ_i.

Now consider the ε-accelerated process $X^\varepsilon(t)$. On each subinterval I_i,

$X_i(u) = X^\varepsilon(t_{i-1} + \varepsilon u)$ is the queue length of a constant $M/M/1$ Queue with arrival rate λ_i, service rate μ_i, and traffic intensity $\rho_i = \lambda_i/\mu_i$. Since $\rho_i < 1$ for all $1 \leq i \leq N$, then each $X_i(u)$ is a stable $M/M/1$ Queue.

Since $X_1(u)$ is a stable $M/M/1$ Queue with a finite initial value X_0, it approaches a steady state as $u \to \infty$, so $X_1(h_1/\varepsilon)$ approaches a steady state as $\varepsilon \to 0$. Thus

$$P[X_1(h_1/\varepsilon) = n] \to \rho_1^n(1 - \rho_1) \text{ as } \varepsilon \to 0.$$

By the Principle of Mathematical Induction, assume

$$P[X_i(h_i/\varepsilon) = n] \to \rho_i^n(1 - \rho_i) \text{ as } \varepsilon \to 0 \text{ for } 1 \leq i \leq k.$$

Then by the Law of Total Probability, we have

$$P[X_{k+1}(h_{k+1}/\varepsilon) = n] = \sum_{j=0}^{\infty} P[X_{k+1}(h_{k+1}/\varepsilon) = n \mid X_{k+1}(0) = j]P[X_{k+1}(0) = j].$$

But $X_{k+1}(0) = X^\varepsilon(t_k) = X^\varepsilon(t_{k-1} + \varepsilon(h_k/\varepsilon)) = X_k(h_k/\varepsilon)$, so

$$P[X_{k+1}(h_{k+1}/\varepsilon) = n] = \sum_{j=0}^{\infty} P[X_{k+1}(h_{k+1}/\varepsilon) = n \mid X_k(h_k/\varepsilon) = j]P[X_k(h_k/\varepsilon) = j].$$

Let $P_j[A] = P[A \mid X_k(h_k/\varepsilon) = j]$ for any event A, then

$$\begin{aligned} P[X_{k+1}(h_{k+1}/\varepsilon) = n] &= \sum_{j=0}^{m} P_j[X_{k+1}(h_{k+1}/\varepsilon) = n]P[X_k(h_k/\varepsilon) = j] \\ &\quad + \sum_{j=m+1}^{\infty} P_j[X_{k+1}(h_{k+1}/\varepsilon) = n]P[X_k(h_k/\varepsilon) = j]. \end{aligned}$$

If we consider $\varepsilon \to 0$, the partial sum formula of a geometric series implies,

$$\begin{aligned}
\limsup_{\varepsilon\to 0} P[X_{k+1}(h_{k+1}/\varepsilon) = n] &= \limsup_{\varepsilon\to 0} \Big(\sum_{j=0}^{m} P_j[X_{k+1}(h_{k+1}/\varepsilon) = n] P[X_k(h_k/\varepsilon) = j] \\
&\quad + \sum_{j=m+1}^{\infty} P_j[X_{k+1}(h_{k+1}/\varepsilon) = n] P[X_k(h_k/\varepsilon) = j] \Big) \\
&= \rho_{k+1}^n (1-\rho_{k+1})(1-\rho_k^{m+1}) \\
&\quad + \limsup_{\varepsilon\to 0} \Big(\sum_{j=m+1}^{\infty} P_j[X_{k+1}(h_{k+1}/\varepsilon) = n] P[X_k(h_k/\varepsilon) = j] \Big)
\end{aligned}$$

And since $0 \le P[A] \le 1$ for any event A,

$$\limsup_{\varepsilon\to 0} P[X_{k+1}(h_{k+1}/\varepsilon) = n] \le \rho_{k+1}^n (1-\rho_{k+1})(1-\rho_k^{m+1}) + \limsup_{\varepsilon\to 0} \Big(\sum_{j=m+1}^{\infty} P[X_k(h_k/\varepsilon) = j] \Big)$$

$$\Rightarrow \limsup_{\varepsilon\to 0} P[X_{k+1}(h_{k+1}/\varepsilon) = n] \le \rho_{k+1}^n (1-\rho_{k+1})(1-\rho_k^{m+1}) + \limsup_{\varepsilon\to 0} P[X_k(h_k/\varepsilon) \ge m+1]$$

$$\Rightarrow \limsup_{\varepsilon\to 0} P[X_{k+1}(h_{k+1}/\varepsilon) = n] \le \rho_{k+1}^n (1-\rho_{k+1})(1-\rho_k^{m+1}) + \limsup_{\varepsilon\to 0} (1 - P[X_k(h_k/\varepsilon) \le m])$$

$$\Rightarrow \limsup_{\varepsilon\to 0} P[X_{k+1}(h_{k+1}/\varepsilon) = n] \le \rho_{k+1}^n (1-\rho_{k+1})(1-\rho_k^{m+1}) + \rho_k^{m+1}.$$

Let $m \to \infty$, then

$$\limsup_{\varepsilon\to 0} P[X_{k+1}(h_{k+1}/\varepsilon) = n] \le \rho_{k+1}^n (1-\rho_{k+1}).$$

Additionally, as $\varepsilon \to 0$, we have

$$\begin{aligned}\liminf_{\varepsilon\to 0} P[X_{k+1}(h_{k+1}/\varepsilon) = n] &= \liminf_{\varepsilon\to 0} \Big(\sum_{j=0}^{m} P_j[X_{k+1}(h_{k+1}/\varepsilon) = n] P[X_k(h_k/\varepsilon) = j] \\ &\quad + \sum_{j=m+1}^{\infty} P_j[X_{k+1}(h_{k+1}/\varepsilon) = n] P[X_k(h_k/\varepsilon) = j] \Big)\end{aligned}$$

$$\Rightarrow \liminf_{\varepsilon\to 0} P[X_{k+1}(h_{k+1}/\varepsilon) = n] \geq \rho_{k+1}^n (1-\rho_{k+1})(1-\rho_k^{m+1}).$$

But as $m \to \infty$, $\rho_k^{m+1} \to 0$. Thus as $m \to \infty$

$$\Rightarrow \limsup_{\varepsilon\to 0} P[X_{k+1}(h_{k+1}/\varepsilon) = n] \leq \rho_{k+1}^n (1-\rho_{k+1})$$

and

$$\Rightarrow \liminf_{\varepsilon\to 0} P[X_{k+1}(h_{k+1}/\varepsilon) = n] \geq \rho_{k+1}^n (1-\rho_{k+1}).$$

Then

$$\rho_{k+1}^n(1-\rho_{k+1}) \leq \liminf_{\varepsilon\to 0} P[X_{k+1}(h_{k+1}/\varepsilon) = n] \leq \limsup_{\varepsilon\to 0} P[X_{k+1}(h_{k+1}/\varepsilon) = n] \leq \rho_{k+1}^n(1-\rho_{k+1})$$

$$\Rightarrow P[X_{k+1}(h_{k+1}/\varepsilon) = n] \to \rho_{k+1}^n(1-\rho_{k+1}) \text{ as } \varepsilon \to 0$$

$$\Rightarrow P[X^\varepsilon(t_{k+1}) = n] \to \rho_{k+1}^n(1-\rho_{k+1}) \text{ as } \varepsilon \to 0$$

$$\Rightarrow P[X^\varepsilon(t) = n] = P[X^\varepsilon(t_N) = n] \to \rho_N^n(1-\rho_N) = (\rho(t))^n(1-\rho(t)) \text{ as } \varepsilon \to 0.$$

Recall from Section 2.3 that for a stable $M/M/1$ Queue in steady state, $P[\tau < t] \to 1$ as $t \to \infty$, where τ is the first time when the queue length is zero. Then we have

$$P[X_N(u) = 0, \text{ for some } u \in (0, h_N/\varepsilon)] = P[\tau < h_N/\varepsilon] \to 1 \text{ as } \varepsilon \to 0$$

$$\Rightarrow P[\Omega_\varepsilon] = P[X^\varepsilon(u) = 0, \text{ for some } u \in (t - h_N, t)] \to 1 \text{ as } \varepsilon \to 0.$$

Theorem 2 *If λ and μ are left-continuous positive step functions and $\rho^*(t) < 1$, then*

$$P[X^\varepsilon(t) = n] \to (\rho(t))^n(1 - \rho(t)) \ , \ \textit{as } \varepsilon \to 0.$$

There exists a partition Δ of $[t_0, t]$ with $t_0 < t_1 < \cdots < t_N = t$ such that $\lambda = \lambda_i$ and $\mu = \mu_i$ for constant λ_i and μ_i on the subinterval $I_i = (t_{i-1}, t_i]$ of width $h_i = t_i - t_{i-1}$ for $1 \le i \le N$. We can choose Δ such that $h_N > 0$ is small enough so that $\rho^*(t_{N-1}) < 1$. Since $\rho^*(t_{N-1}) < 1$, then $\lambda_{N-1} < \mu_{N-1}$. Then there is some $\delta > 0$ such that

$$\rho^*(t_{N-1}) < 1 - \delta$$

$$\Leftrightarrow \int_v^{t_{N-1}} \lambda(s)\, ds < (1 - \delta) \int_v^{t_{N-1}} \mu(s)\, ds \ , \text{ for all } v \in [t_0, t_{N-1})$$

$$\Leftrightarrow \sum_{j=i}^{N-1} \lambda_j h_j < (1 - \delta) \sum_{j=i}^{N-1} \mu_j h_j \ , \text{ for all } 1 \le i < N - 1. \tag{3.1}$$

Now we will define a new arrival rate function $\widetilde{\lambda}(t)$ as follows:

First, let $\widetilde{\lambda} = \lambda$, and start at $I_1 = [t_0, t_1]$. If $\widetilde{\lambda}_1 < \mu_1$, there is no change; move on to I_2. If $\widetilde{\lambda}_1 \ge \mu_1$, then change $\widetilde{\lambda_1}$ to $(1 - \delta)\mu_1$. So we reduced the area below $\widetilde{\lambda}$ over I_1 by $A_1 = [\lambda_1 - (1 - \delta)\mu_1]h_1$. Now increase $\widetilde{\lambda}_2$ so the area below $\widetilde{\lambda}$ over I_2 is increased by A_1.

After this change, $\widetilde{\lambda}_1 < \mu_1$ and (3.1) still holds for $i \geq 2$, since

$$(3.1) \Longleftrightarrow [\lambda_1 - (1-\delta)\mu_1]h_1 + \sum_{j=2}^{N-1} \lambda_j h_j < (1-\delta) \sum_{j=2}^{N-1} \mu_j h_j$$

In terms of the arrival process, what we have done is to take a Poisson number of arrivals of mean A_1 from I_1 and delay them to I_2. This could only possibly make the system more congested at time t, since the server has less time to catch up when the arrivals come later.

Now move on to I_2 and repeat what we have done for $\widetilde{\lambda}_1$ and $\widetilde{\lambda}_2$ with $\widetilde{\lambda}_2$ and $\widetilde{\lambda}_3$. Continue in this way. When we reach I_{N-2}, we have obtained a new system that has arrival rate $\widetilde{\lambda}(t)$ and service rate $\mu(t)$, with $\widetilde{\lambda}_i < \mu_i$ for all $1 \leq i \leq n$. Denote it's queue length by $\widetilde{X}(t)$. A simple example of this process is shown in Figure 3.1.

Now look at the ε-accelerated process $\widetilde{X}^{\varepsilon}(t)$. By Theorem 1, $P[\widetilde{\Omega}_{\varepsilon}] \to 1$ as $\varepsilon \to 0$ where $\widetilde{\Omega}_{\varepsilon}$ is the event that $\widetilde{X}^{\varepsilon}(s) = 0$ for some $s \in (t_{N-1}, t)$. Which means at some time in the subinterval I_N, as $\varepsilon \to 0$, $\widetilde{X}^{\varepsilon}(t) = 0$ and the queue is empty.

On $\widetilde{\Omega}_{\varepsilon}$, the queue is empty at some time $s \in (t_{N-1}, t)$. We now reverse the process that we used to create $\widetilde{\lambda}$. At each step, we are advancing arrivals; giving the server more time to serve them. So the queue is still empty at time s, therefore we still have $\widetilde{\Omega}_{\varepsilon}$, and we didn't change the system after s. When we are finished with these steps, we obtain our original arrival rate, λ. Thus we have our original queue length process ε-accelerated, and we still have $\widetilde{\Omega}_{\varepsilon}$; therefore

$$X^{\varepsilon}(t) = \widetilde{X}^{\varepsilon}(t) \text{ on } \widetilde{\Omega}_{\varepsilon}.$$

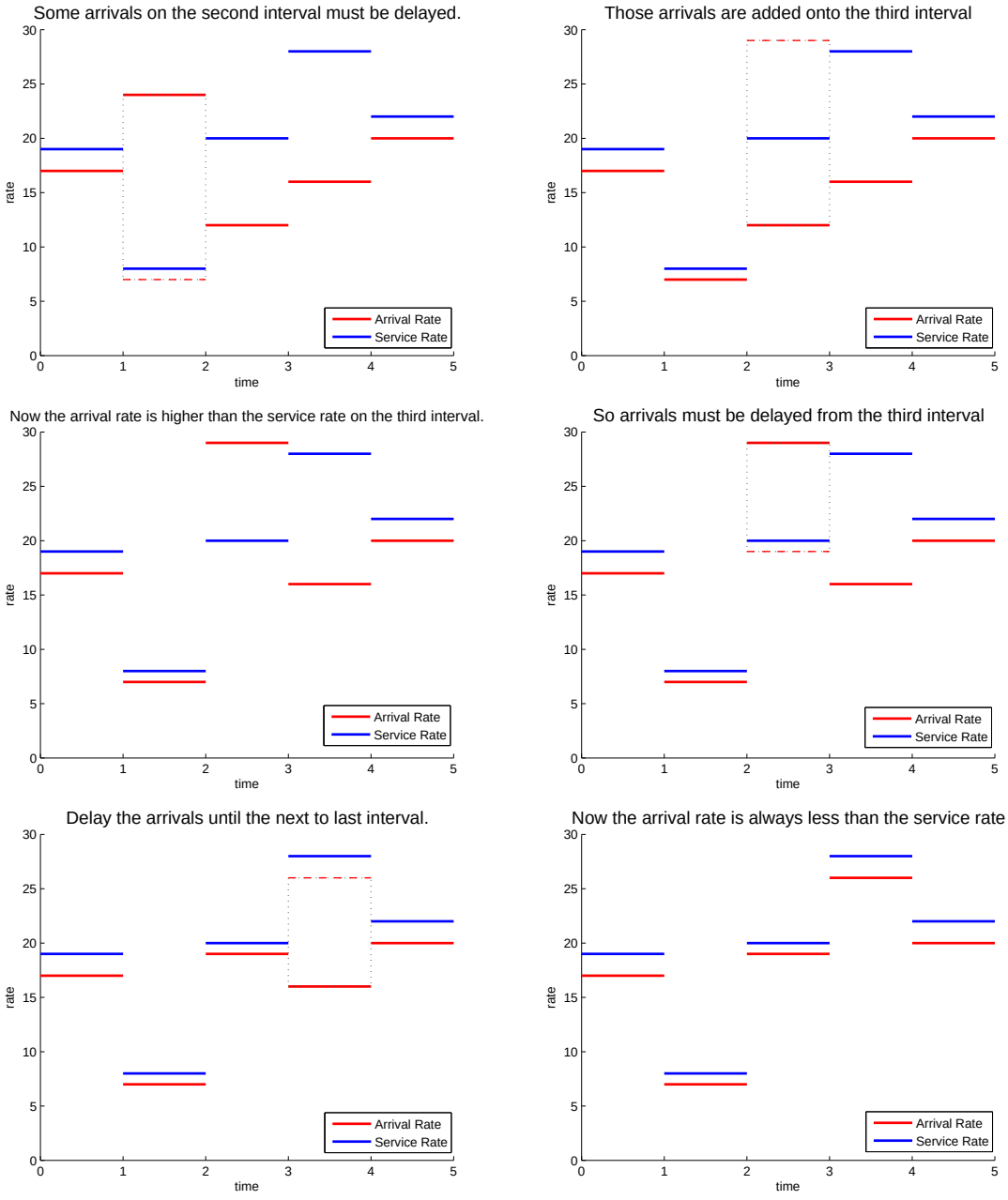

Figure 3.1: An Example of Delaying Arrivals

We have

$$\begin{aligned} P[X^\varepsilon(t)=n] &= P[X^\varepsilon(t)=n;\ \widetilde{\Omega}_\varepsilon] + P[X^\varepsilon(t)=n;\ \widetilde{\Omega}_\varepsilon^C] \\ &= P[\widetilde{X}^\varepsilon(t)=n;\ \widetilde{\Omega}_\varepsilon] + P[X^\varepsilon(t)=n;\ \widetilde{\Omega}_\varepsilon^C] \\ &= P[\widetilde{X}^\varepsilon(t)=n] - P[\widetilde{X}^\varepsilon(t)=n;\ \widetilde{\Omega}_\varepsilon^C] + P[X^\varepsilon(t)=n;\ \widetilde{\Omega}_\varepsilon^C]. \end{aligned}$$

But by Theorem 1,

$$P[\widetilde{\Omega}_\varepsilon^C] \to 0 \ \text{ and } \ P[\widetilde{X}^\varepsilon(t)=n] \to (\rho(t))^n(1-\rho(t)) \ \text{ as } \varepsilon \to 0.$$

Which implies that

$$P[X^\varepsilon(t)=n] \to (\rho(t))^n(1-\rho(t)) \ \text{ as } \varepsilon \to 0.$$

Theorem 3 *Let λ and μ be piecewise continuous on $[t_0,t]$, and assume $\rho^*(t)<1$. Then*

$$P[X^\varepsilon(t)\geq n] \to (\rho(t))^n \ \textit{as } \varepsilon \to 0 \ \textit{, and hence}$$

$$P[X^\varepsilon(t)=n] \to (\rho(t))^n(1-\rho(t)) \ \textit{as } \varepsilon \to 0.$$

First, we can approximate λ and μ arbitrarily closely by step functions $\lambda^+(t)$ and $\mu^-(t)$ such that $\lambda^+(s)\geq\lambda(s)$ and $\mu^-(s)\leq\mu(s)$ for all $s\in[t_0,t]$ and $\rho^*(t;\lambda^+,\mu^-)<1$.[1] Consider the $M(t)/M(t)/1$ Queue $X^+(t)$ with initial state $X^+(t_0)=X_0$, arrival rate $\lambda^+(t)$, and service rate $\mu^-(t)$. Then $\rho^+(t)=\lambda^+(t)/\mu^-(t)$ and $\rho^{+*}(t):=\rho^*(t;\lambda^+,\mu^-)<1$. Note that

[1] Proof in Appendix A

by the Stochastic Dominance Principle and our choice of λ^+ and μ^-,

$P[X^\varepsilon(t) \geq n] \leq P[X^{+\varepsilon}(t) \geq n]$ for the ε-accelerated processes X^ε and $X^{+\varepsilon}$. By Theorem 2, we have

$$P[X^\varepsilon(t) \geq n] \leq P[X^{+\varepsilon}(t) \geq n] \to (\rho^+(t))^n \text{ as } \varepsilon \to 0.$$

Similarly, we can choose $\lambda^-(t)$ and $\mu^+(t)$ such that $\lambda^-(s) \leq \lambda(s)$ and $\mu^+(s) \geq \mu(s)$ for all $s \in [t_0, t]$ and $\rho^*(t; \lambda^-, \mu^+) < 1$, and consider the $M(t)/M(t)/1$ Queue $X^-(t)$ with initial state $X^-(t_0) = X_0$, arrival rate $\lambda^-(t)$, and service rate $\mu^+(t)$. Here, $\rho^-(t) = \lambda^-(t)/\mu^+(t)$ and $\rho^{-*}(t) := \rho^*(t; \lambda^-, \mu^+) < 1$. Note that by the Stochastic Dominance Principle and our choice of λ^- and μ^+, $P[X^\varepsilon(t) \geq n] \geq P[X^{-\varepsilon}(t) \geq n]$ for the ε-accelerated processes X^ε and $X^{-\varepsilon}$. So, again by Theorem 2, we have

$$P[X^\varepsilon(t) \geq n] \geq P[X^{-\varepsilon}(t) \geq n] \to (\rho^-(t))^n \text{ as } \varepsilon \to 0.$$

Since we can choose $\lambda^+(t)$ and $\mu^-(t)$ arbitrarily close to λ and μ, then $\rho^+(t)$ can be made arbitrarily close to $\rho(t)$. Similarly, $\rho^-(t)$ can be made arbitrarily close to $\rho(t)$. Thus

$$(\rho^-(t))^n \leftarrow P[X^{-\varepsilon}(t) \geq n] \leq P[X^\varepsilon(t) \geq n] \leq P[X^{+\varepsilon}(t) \geq n] \to (\rho^+(t))^n \text{ as } \varepsilon \to 0$$

$$\implies P[X^\varepsilon(t) \geq n] \to (\rho(t))^n \text{ as } \varepsilon \to 0$$

$$\begin{aligned} \implies P[X^\varepsilon(t) = n] &= P[X^\varepsilon(t) \geq n] - P[X^\varepsilon(t) \geq n+1] \to (\rho(t))^n - (\rho(t))^{n+1} \\ &= (\rho(t))^n(1 - \rho(t)) \text{ as } \varepsilon \to 0. \end{aligned}$$

Theorem 4 *If $\rho^*(t) > 1$, then $X^\varepsilon(t) \to \infty$ as $\varepsilon \to 0$, in distribution.*

Assume $\rho^*(t) > 1$, then there exists $v \in [t_0, t)$ such that

$$\int_v^t \lambda(s)\, ds > \int_v^t \mu(s)\, ds.$$

Consider the ε-accelerated system $X^\varepsilon(t)$ with arrival rate $\lambda(t)/\varepsilon$ and service rate $\mu(t)/\varepsilon$, and let $\varepsilon = \frac{1}{n}$.

Then we have a nonhomogeneous poisson arrival process on the interval $(v, t]$, say $A^\varepsilon(t)$, of rate $\lambda(t)/\varepsilon = n\lambda(t)$. And the number of services the server can complete is a nonhomogeneous poisson process, say $S^\varepsilon(t)$ of rate $\mu(t)/\varepsilon = n\mu(t)$. Note that this is not necessarily the actual amount of service done, just the amount possible; the queue may be empty at some point, in which case the server isn't actually doing the work that it could be. This means that in the sense of distribution we can think of these processes at time t as the sum of n i.i.d Poisson random variables, $A^\varepsilon(t) = A_1 + A_2 + \cdots + A_{n-1} + A_n$ and $S^\varepsilon(t) = S_1 + S_2 + \cdots + S_{n-1} + S_n$, where $A_i(t)$ has a mean of $\int_v^t \lambda(s)\, ds$ and $S_i(t)$ has mean $\int_v^t \mu(t)\, ds$ for $1 \le i \le n$. Since $S^\varepsilon(t)$ is the number of services that can be completed, the queue length is at least $A^\varepsilon(t) - S^\varepsilon(t)$. So we have

$$X^\varepsilon(t) \ge \sum_{i=0}^{n} A_i(t) - \sum_{i=0}^{n} S_i(t)$$

$$\Rightarrow \frac{X^\varepsilon(t)}{n} \ge \frac{\sum_{i=0}^{n} A_i(t)}{n} - \frac{\sum_{i=0}^{n} S_i(t)}{n}.$$

By the Strong Law of Large Numbers,

$$\frac{1}{n}\sum_{i=0}^{n} A_i(t) \to E[A_1(t)] \text{ and } \frac{1}{n}\sum_{i=0}^{n} S_i(t) \to E[S_1(t)] \text{ as } n \to \infty.$$

Therefore

$$\lim_{n\to\infty} \frac{X^{\varepsilon}(t)}{n} \geq E[A_1(t)] - E[S_1(t)] = \int_v^t \lambda(s)\, ds - \int_v^t \mu(t)\, ds > 0$$

and

$$X^{\varepsilon}(t) \to \infty \quad \text{as} \quad \varepsilon = \frac{1}{n} \to 0.$$

Chapter 4

Extensions

4.1 $M(t)/M(t)/k(t)$ Queues

Now consider a queuing system with multiple homogeneous servers, where the number of available servers at any time t is given by a positive integer-valued step function $k(t)$. Let $X(t)$ be the queue length process with arrival rate $\lambda(t)$ and with $k(t)$ servers each working at a rate of $\mu(t)$; where $\lambda(t)$ and $\mu(t)$ are piecewise continuous functions of time. The traffic intensity is now $\rho(t) = \lambda(t)/(k(t)\mu(t))$. For the multiple server case, we redefine our stability function as

$$\rho^*(t) := \rho^*(t; \lambda, \mu, k) = \sup_{t_* \in (t_0, t)} \frac{\int_{t_*}^{t} \lambda(s)\, ds}{\int_{t_*}^{t} k(s)\mu(s)\, ds}$$

If $\rho^*(t) > 1$, we again have $X^\varepsilon(t) \to \infty$ as $\varepsilon \to 0$, in distribution. The proof of this is exactly like that of Theorem 4, but using our new ρ^*.

If $\rho^*(t) < 1$, then the limiting probability in the multiple server case is once again the same as it's constant counterpart; the proof only requires a few simple changes. First, we can still approximate λ and μ with step functions to make new processes $X^+(t)$ and $X^-(t)$, each with $\rho^*(t) < 1$, such that $P[X^{-\varepsilon}(t) = n] \leq P[X^\varepsilon(t) = n] \leq P[X^{+\varepsilon}(t) = n]$ as in Theorem 3. Then for each of these processes, as in Theorem 2, we partition the interval $[t_0, t]$ making λ, μ, and k constant on each subinterval I_i. So we have

$$\int_v^{t_{N-1}} \lambda(s)\, ds < (1-\delta) \int_v^{t_{N-1}} k(s)\mu(s)\, ds \text{ , for all } v \in [t_0, t_{N-1})$$

$$\Leftrightarrow \sum_{j=i}^{N-1} \lambda_j h_j < (1-\delta) \sum_{j=i}^{N-1} k_j \mu_j h_j \text{ , for all } 1 \leq i < N-1$$

We then delay arrivals so that $\lambda_i < k_i \mu_i$ on each interval I_i. After ε-accelerating these new processes, on each interval we have an $M/M/k$ queue approaching steady state as $\varepsilon \to 0$. So $P[X^{-\varepsilon}(t_{N-1}) < \infty] \to 1$ as $\varepsilon \to 0$, and the queue is empty at some time in the last interval with probability approaching 1 as $\varepsilon \to 0$; and the same is true for $X^{+\varepsilon}$. Now if we advance arrivals it won't affect the system at time t. Note that this is why we need homogeneous servers; if the servers were different, when advancing arrivals we could end up moving an arrival to a slower server that might not be finished at time t, therefore affecting the distribution of $X^{\varepsilon}(t)$. With homogeneous servers, the distribution of the service times is the same, so we can advance arrivals safely. We then limit the probabilities of $X^{-\varepsilon}(t)$ and $X^{+\varepsilon}(t)$ as in Theorem 3, squeezing them to the probabilities of $X^{\varepsilon}(t)$. So after all this, we get the same limiting probability as the $M/M/k$ queue on the last interval; as $\varepsilon \to 0$

$$P[X^{\varepsilon}(t) = n] \to p_n = \begin{cases} \dfrac{1}{n!} \left(\dfrac{\lambda(t)}{\mu(t)} \right)^n p_0 & \text{for } 1 \leq n \leq k(t) \\ \dfrac{1}{(k(t))^{n-k(t)} k(t)!} \left(\dfrac{\lambda(t)}{\mu(t)} \right)^n p_0 & \text{for } n \geq k(t), \end{cases}$$

where

$$p_0 = \left[\frac{1}{k(t)!} \left(\frac{\lambda(t)}{\mu(t)} \right)^{k(t)} \left(\frac{k(t)\mu(t)}{k(t)\mu(t) - \lambda(t)} \right) + \sum_{n=0}^{k(t)-1} \frac{1}{n!} \left(\frac{\lambda(t)}{\mu(t)} \right)^n \right]^{-1}.$$

4.2 Finite Capacity

For the finite capacity case, there is always a limiting probability, since there are only finitely many states. We don't even need a concept of stability, though ρ^* is still a helpful quantity for the analysis of the queue; if $\rho^* > 1$, then the queue will spend much more time

at capacity or near it. Once again, the limiting probabilities are the same as in the finite case, and the proof consists of simple changes to the previous analysis.

Let $X(t)$ be the length of a queue with a maximum capacity of N, arrival rate $\lambda(t)$, and $k(t)$ servers each working at a rate of $\mu(t)$; where $\lambda(t)$ and $\mu(t)$ are piecewise continuous functions of time and $k(t)$ is a positive integer-valued step function. Then we call $X(t)$ an $M(t)/M(t)/k(t)/N$ queue. To find the steady state distribution of $X(t)$, we once again ε-accelerate the process. While it may seem problematic to let the arrival rate approach infinity when we have a finite capacity, it is not, since we also let the service rate approach infinity at the same pace. As in Theorem 3, approximate λ and μ with step functions to make the processes $X^+(t)$ and $X^-(t)$ such that $P[X^{-\varepsilon}(t) = n] \leq P[X^{\varepsilon}(t) = n] \leq P[X^{+\varepsilon}(t) = n]$. Then on each interval I_i these processes are a constant $M/M/k/N$ queue with arrival rate λ_i, service rate μ_i, k_i servers, and capacity N. As we ε-accelerate these processes, the constant $M/M/k/N$ queues approach steady state. We then limit our approximations of λ and μ and squeeze their probabilities to the $M/M/k/N$ queue on the last interval, with arrival rate $\lambda(t)/\varepsilon$, $k(t)$ servers with rate rate $\mu(t)/\varepsilon$, and capacity N. So as $\varepsilon \to 0$,

$$P[X^{\varepsilon}(t) = n] \to p_n = \begin{cases} \dfrac{1}{n!}\left(\dfrac{\lambda(t)}{\mu(t)}\right)^n p_0 & \text{for } 1 \leq n \leq k(t) \\ \dfrac{1}{k(t)^{n-k(t)}k(t)!}\left(\dfrac{\lambda(t)}{\mu(t)}\right)^n p_0 & \text{for } k(t) \leq n \leq N, \end{cases}$$

where

$$p_0 = \begin{cases} \left[\dfrac{1}{k(t)!}\left(\dfrac{\lambda(t)}{\mu(t)}\right)^{k(t)} \dfrac{1-\left(\frac{\lambda(t)}{k(t)\mu(t)}\right)^{N-k+1}}{1-\frac{\lambda(t)}{k(t)\mu(t)}} + \displaystyle\sum_{n=0}^{k(t)-1} \frac{1}{n!}\left(\frac{\lambda(t)}{\mu(t)}\right)^n\right]^{-1} & \text{for } \dfrac{\lambda(t)}{k(t)\mu(t)} \neq 1 \\ \left[\dfrac{1}{k(t)!}\left(\dfrac{\lambda(t)}{\mu(t)}\right)^{k(t)} (N-k(t)+1) + \displaystyle\sum_{n=0}^{k(t)-1} \frac{1}{n!}\left(\frac{\lambda(t)}{\mu(t)}\right)^n\right]^{-1} & \text{for } \dfrac{\lambda(t)}{k(t)\mu(t)} = 1. \end{cases}$$

Chapter 5

Simulation

In this final chapter, we model a time dependent queuing system using data from an Auburn University web server. The arrival process for our model is taken from the server's logs. The number of requests per hour from five consecutive Wednesdays was averaged and used to make an arrival process for our queue. For an $M/M/1$ Queue in steady state, the distribution of the departure process is the same as the arrival process, so we assume this approximation will be a valid model of the arrival process whether the logs record the request at the time of arrival or after the request has been completed. The service rate in our model is kept constant for a more controlled analysis of the data, which is reasonable since a computer can usually work at a close to steady rate. The service rate of the actual web server will most likely be much higher, and therefore avoid huge backups, but then we could not show what happens when $\rho^* \geq 1$. The reason this approach to studying an $M(t)/M(t)/1$ queue is so applicable to a web server is that such a server already has a very high arrival and service rate, and can essentially be considered ε-accelerated without actually changing the rates. In this model, we assume that individual arrivals to a server are independent exponential random variables, as is each request made by those arrivals. We also assume the server has some way of keeping arrivals in a queue, that it does not simply reject another arrival when it is busy. This analysis will show the importance of ρ^* as a stability condition. Figure 5.1 illustrates the service and arrival rates of this system and $\rho^*(t)$ at every time t.

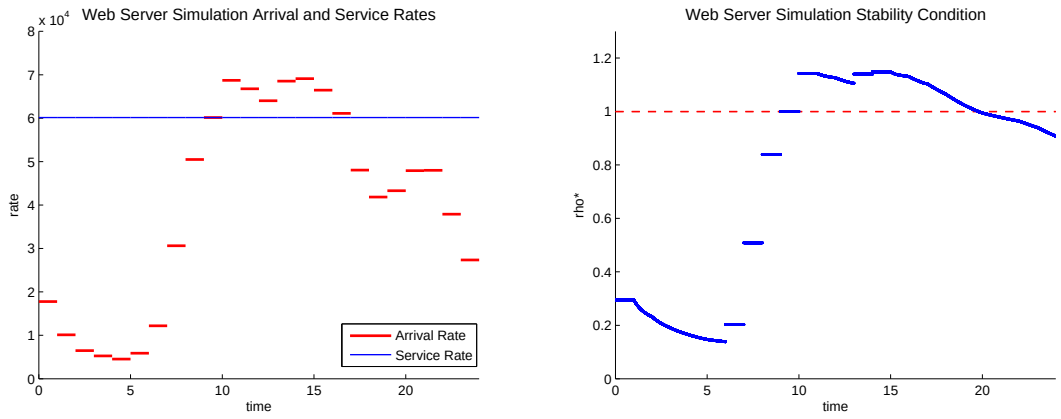

Figure 5.1: Arrival and Service rates and $\rho^*(t)$

In order to simulate the queue, we use step functions for the arrival and service rate function. Then on each interval, we simulate an $M/M/1$ queue by simulating exponential random variables for the interarrival times and required services. One can simulate any continuous random variable X with distribution function $F(x)$ by generating a random number U, which is uniformly distributed on $(0, 1)$, and setting $X = F^{-1}(U)$, as shown in Chapter 7 of Solomon [5]. Thus, we simulate our exponential random variables by letting $X = \frac{1}{\beta} \ln U$, where β is the rate parameter of the random variable and U is a random number uniformly distributed on $(0, 1)$.

First consider Figure 5.2, the case where $\rho^*(t) < 1$ and the queue is stable. Notice where the queue length climbs to a point that makes the length of the queue at earlier times seem insignificant. This is when $\rho^*(t) > 1$; but when $\rho^*(t) < 1$, the server catches up and brings the queue back down to manageable size. You can even notice that the queue seems to jump around quite a bit right before time 10. This is where $\rho^*(t) = 1$, or close enough so that we can consider it 1.

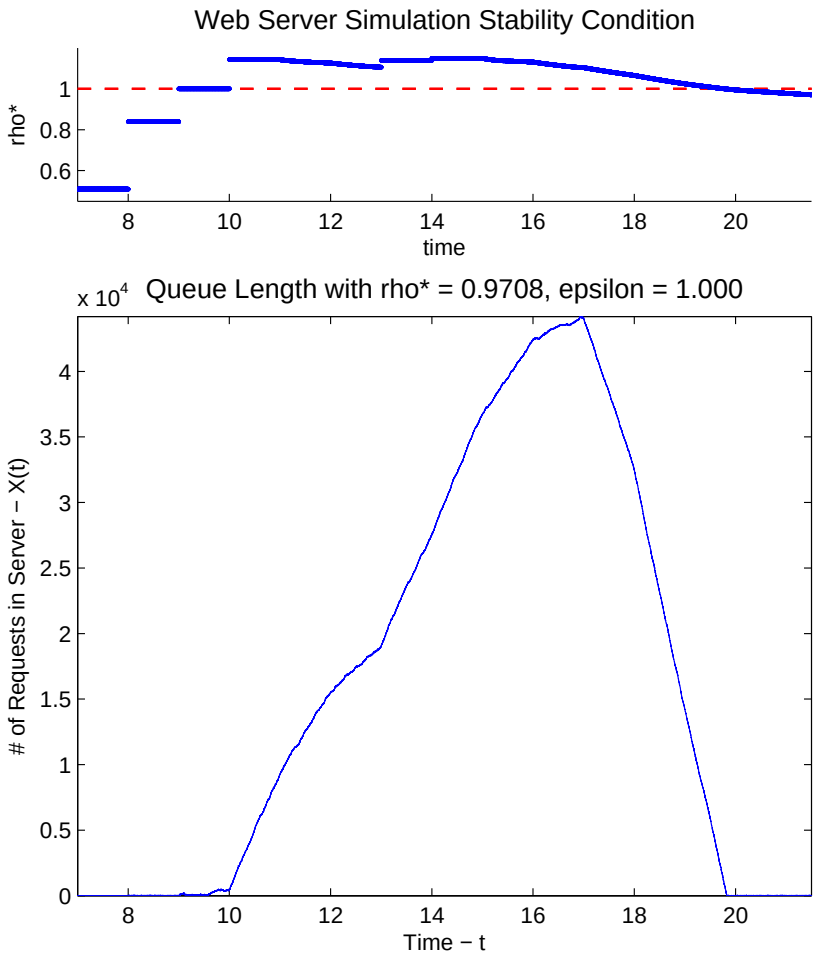

Figure 5.2: $\rho^*(t) < 1$ without ε-acceleration

With epsilon = 1.0000000, the system cleared out
at 21.00155 after having a maximum queue length
of 43555. Ran 200 times without acceleration.

Calculated Long Run Fractions of Time / Probabilities
That is, P[X = n] for each state below.

n	Theoretical	Accelerated	Multiple Runs
0	0.20213661	0.19759187	0.24000000
1	0.16127740	0.15603091	0.15500000
2	0.12867733	0.12492567	0.10000000
3	0.10266693	0.10043291	0.10500000
4	0.08191419	0.08225265	0.08500000
5	0.06535633	0.06666348	0.07500000
6	0.05214542	0.05503601	0.04500000
7	0.04160493	0.04583023	0.03500000
8	0.03319505	0.03707380	0.00500000
9	0.02648511	0.02673086	0.02500000
10	0.02113150	0.01998091	0.03000000
11	0.01686005	0.01477285	0.01000000
12	0.01345202	0.01334929	0.01500000
13	0.01073287	0.01170998	0.02500000
14	0.00856337	0.00924480	0.01000000
15	0.00683240	0.00708840	0.01000000
16	0.00545132	0.00567585	0.01000000
17	0.00434941	0.00510566	0.00000000
18	0.00347023	0.00446735	0.00000000
19	0.00276877	0.00395070	0.00000000

Figure 5.3: Calculated Probabilities without ε-acceleration

Figure 5.3 shows the theoretical probability $\lim_{\varepsilon \to 0} P[X^{\varepsilon}(t) = n]$ for $n = 0$ to $n = 19$, the fraction of time the simulated queue is in these first 20 states after clearing itself out on the last interval, and the fraction of times these states occurred when the system was run multiple times.

We can now see in Figures 5.4 and 5.5 that the ε-acceleration does not drastically change the calculated probabilities nor the shape of the graph.

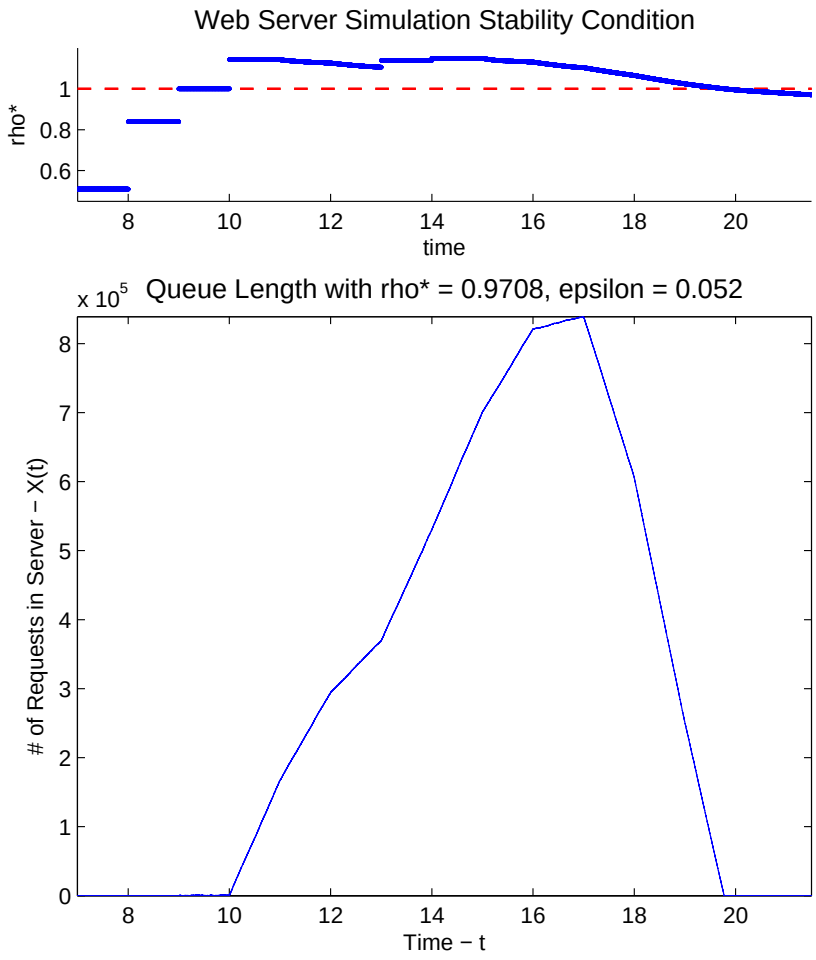

Figure 5.4: ε-accelerated with $\rho^*(t) < 1$

```
With epsilon = 0.0077808, the system cleared out
at 21.00002 after having a maximum queue length
of 5617345.  Ran 200 times without acceleration.

Calculated Long Run Fractions of Time / Probabilities
That is, P[X = n] for each state below.
n       Theoretical         Accelerated     Multiple Runs
0       0.20213661          0.20254598      0.21500000
1       0.16127740          0.16204405      0.15000000
2       0.12867733          0.12932871      0.11500000
3       0.10266693          0.10298326      0.13500000
4       0.08191419          0.08213674      0.04500000
5       0.06535633          0.06534474      0.07500000
6       0.05214542          0.05210977      0.07000000
7       0.04160493          0.04149426      0.03500000
8       0.03319505          0.03288973      0.03000000
9       0.02648511          0.02620844      0.02500000
10      0.02113150          0.02083914      0.02000000
11      0.01686005          0.01665043      0.01500000
12      0.01345202          0.01322316      0.01500000
13      0.01073287          0.01064279      0.01500000
14      0.00856337          0.00850770      0.00000000
15      0.00683240          0.00668275      0.01000000
16      0.00545132          0.00532815      0.00500000
17      0.00434941          0.00431330      0.00000000
18      0.00347023          0.00340835      0.00500000
19      0.00276877          0.00270998      0.00000000
```

Figure 5.5: Calculated Probabilities with ε-acceleration

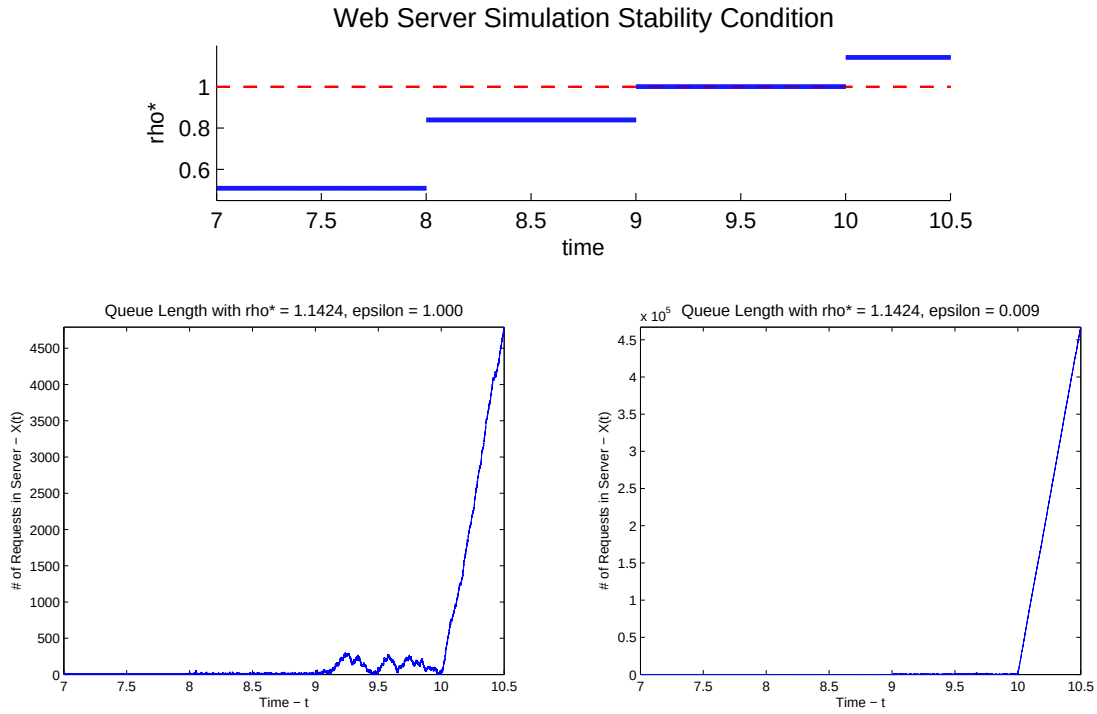

Figure 5.6: $\rho^*(t) > 1$

For $\rho^*(t) > 1$ we can see that the queue length is getting arbitrarily large. In Figure 5.6, once again the ε-accelerated version looks the same as the original system, due to it's high arrival and service rates. You can also see that on the interval from 9 to 10, where $\rho^*(t) = 1$, the queue is very erratic.

When $\rho^*(t) = 1$, or as close as we can get it numerically, the queue length varies wildly. It is very erratic and unpredictable, as evidenced by the four graphs in Figure 5.7. In Figure 5.8, we see the same phenomenon for the ε-accelerated case.

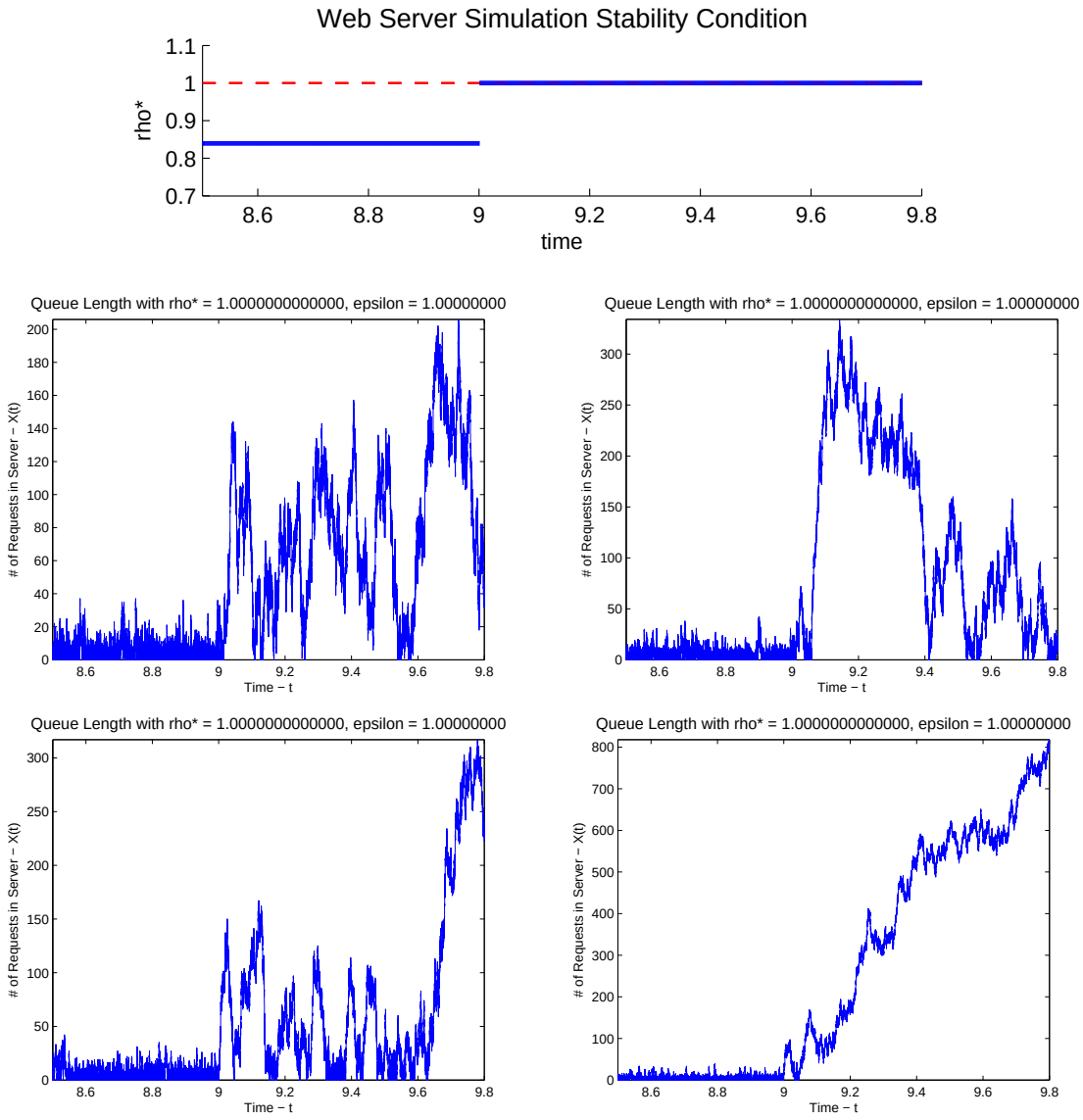

Figure 5.7: $\rho^*(t) = 1$ without ε-acceleration

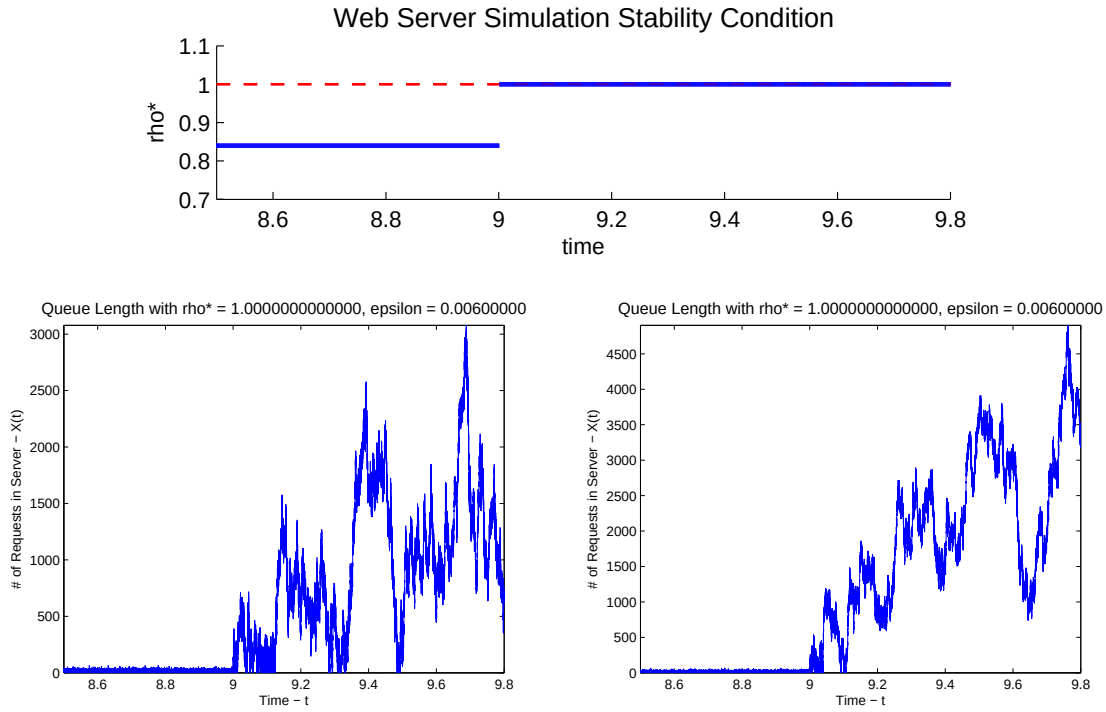

Figure 5.8: ε-accelerated with $\rho^*(t) = 1$

Chapter 6

Conclusion

We have shown that under the special stability function $\rho^*(t)$, the distribution of the queue length of a ε-accelerated $M(t)/M(t)/1$ queue is the same as a queue length of a constant rate $M/M/1$ queue in a steady state. That is, when $\rho^*(t) < 1$,

$$P[X^\varepsilon(t) = n] \to (\rho(t))^n(1 - \rho(t)) \text{ as } \varepsilon \to 0;$$

and when $\rho^*(t) > 1$,

$$X^\varepsilon(t) \to \infty \quad \text{as} \quad \varepsilon \to 0.$$

We did not consider the case when $\rho^*(t) = 1$, though an analysis of it can be found in Massey's work [3]. We were able to extend our results to time dependent queueing systems with multiple homogeneous servers and finite capacities, which is not presented by Massey. Our simulation clearly supports our theory, shows an applied use of this paper, and even gives some empirical insight into the case with $\rho^*(t) = 1$.

Bibliography

[1] Patrick Billingsley, "Probability and Measure," Second Edition, Wiley, 1986.

[2] Donald Gross and Carl M. Harris, "Fundamentals of Queueing Theory," Second Edition, Wiley, 1985.

[3] William Massey, "Asymptotic Analysis of the Time Dependent M/M/1 Queue," Mathematics of Operations Research, Vol. 10, No. 2, 1985.

[4] Sheldon M. Ross, "Stochastic Processes," Wiley, 1983.

[5] Frederick Solomon, "Probability and Stochastic Processes," Prentice-Hall, 1987.

Appendices

Appendix A

Approximating Rates with Step Functions

Let $\lambda(t)$ and $\mu(t)$ be left-continuous positive piecewise functions of time on the interval $[t_0, t]$ with $\rho^*(t; \lambda, \mu) < 1$. Then there exists positive step functions λ^+, λ^-, μ^+, and μ^- such that $\lambda^-(s) \leq \lambda(s) \leq \lambda^+(s)$ and $\mu^-(s) \leq \mu(s) \leq \mu^+(s)$ for all $s \in [t_0, t]$, and $\rho^*(t; \lambda^-, \mu^+) < 1$ and $\rho^*(t; \lambda^-, \mu^+) < 1$.

This is obviously true for λ^- and μ^+, since then $\rho^*(t; \lambda^-, \mu^+) \leq \rho^*(t; \lambda, \mu) < 1$. The other case requires a bit more work. Since μ is positive, then its average value is positive. Hence there is some $\delta_0 > 0$ such that

$$\frac{1}{t - t_*} \int_{t_*}^{t} \mu(s)\, ds > \delta_0 \text{ , for all } t_* \in [t_0, t).$$

Note that

$$\rho^*(t; \lambda, \mu) = \sup_{t_* \in (t_0, t)} \frac{\int_{t_*}^{t} \lambda(s)\, ds}{\int_{t_*}^{t} \mu(s)\, ds} = \sup_{t_* \in (t_0, t)} \frac{\frac{1}{t - t_*} \int_{t_*}^{t} \lambda(s)\, ds}{\frac{1}{t - t_*} \int_{t_*}^{t} \mu(s)\, ds}.$$

Since $\rho^*(t; \lambda, \mu) < 1$, we can choose $0 < \delta_1 < \delta_0$ such that

$$\rho^*(t; \lambda, \mu) + \frac{\delta_1}{\delta_0} < 1,$$

and pick λ^+ such that

$$\sup_{t_* \in (t_0, t)} (\lambda^+(t_*) - \lambda(t_*)) < \delta_1.$$

Then

$$\begin{aligned}
\rho^*(t;\lambda^+,\mu) &= \sup_{t_*\in(t_0,t)} \frac{\frac{1}{t-t_*}\int_{t_*}^t \lambda^+(s)\,ds}{\frac{1}{t-t_*}\int_{t_*}^t \mu(s)\,ds} \le \sup_{t_*\in(t_0,t)} \frac{\frac{1}{t-t_*}\int_{t_*}^t \lambda(s)\,ds+\delta_1}{\frac{1}{t-t_*}\int_{t_*}^t \mu(s)\,ds} \\
&\le \sup_{t_*\in(t_0,t)} \frac{\frac{1}{t-t_*}\int_{t_*}^t \lambda(s)\,ds}{\frac{1}{t-t_*}\int_{t_*}^t \mu(s)\,ds} + \frac{\delta_1}{\frac{1}{t-t_*}\int_{t_*}^t \mu(s)\,ds} \le \sup_{t_*\in(t_0,t)} \frac{\frac{1}{t-t_*}\int_{t_*}^t \lambda(s)\,ds}{\frac{1}{t-t_*}\int_{t_*}^t \mu(s)\,ds} + \frac{\delta_1}{\delta_0} \\
&\le \rho^*(t;\lambda,\mu) + \frac{\delta_1}{\delta_0} < 1.
\end{aligned}$$

Now choose $\delta_2 > 0$ so that

$$\rho^*(t;\lambda^+,\mu) < 1 - \frac{\delta_2}{\delta_0},$$

then

$$\rho^*(t;\lambda^+,\mu)\left(\frac{1}{1-\frac{\delta_2}{\delta_0}}\right) < 1;$$

so choose μ^- so that

$$\sup_{t_*\in(t_0,t)} (\mu(t_*) - \mu^-(t_*)) < \delta_2.$$

Now

$$\begin{aligned}
\rho^*(t;\lambda^+,\mu^-) &= \sup_{t_*\in(t_0,t)} \frac{\frac{1}{t-t_*}\int_{t_*}^t \lambda^+(s)\,ds}{\frac{1}{t-t_*}\int_{t_*}^t \mu^-(s)\,ds} \le \sup_{t_*\in(t_0,t)} \frac{\frac{1}{t-t_*}\int_{t_*}^t \lambda^+(s)\,ds}{\frac{1}{t-t_*}\int_{t_*}^t \mu(s)\,ds - \delta_2} \\
&\le \sup_{t_*\in(t_0,t)} \frac{\frac{1}{t-t_*}\int_{t_*}^t \lambda^+(s)\,ds}{\left(\frac{1}{t-t_*}\int_{t_*}^t \mu(s)\,ds\right)\left[1 - \frac{\delta_2}{\frac{1}{t-t_*}\int_{t_*}^t \mu(s)\,ds}\right]} \\
&\le \sup_{t_*\in(t_0,t)} \frac{\frac{1}{t-t_*}\int_{t_*}^t \lambda^+(s)\,ds}{\left(\frac{1}{t-t_*}\int_{t_*}^t \mu(s)\,ds\right)\left[1 - \frac{\delta_2}{\delta_0}\right]} \\
&\le \left(\sup_{t_*\in(t_0,t)} \frac{\frac{1}{t-t_*}\int_{t_*}^t \lambda^+(s)\,ds}{\frac{1}{t-t_*}\int_{t_*}^t \mu(s)\,ds}\right)\frac{1}{1-\frac{\delta_2}{\delta_0}} = \rho^*(t;\lambda^+,\mu)\left(\frac{1}{1-\frac{\delta_2}{\delta_0}}\right) \\
&< 1.
\end{aligned}$$

Appendix B

Simulation Code

```
% queuingsystemgraph.m
% Allen Flick Master's Thesis Queuing System Simulation
% Simulates and Plots a Queuing system as follows:
% Customers arrive according to a non-homogeneous poisson process
% of rate 'arate', and are served with rate 'srate', both of which
% are piecewise constant functions of time.
function queuingsystemgraph(initialSize, arate, srate, timeint,...
    startTime, endTime, epsilon)
%default values
if nargin < 1
    arate=[7.5,3];
    srate=[5,6];
    timeint=[0,1,2];
    startTime=timeint(1);
    endTime=timeint(end);
    initialSize=5;
    epsilon=1;
end
etol=.0000001;
j=1;
```

```
while (timeint(j)<startTime-etol), j=j+1; end
if timeint(j)-startTime>etol, j=j-1; end
lastInt=length(timeint)-1;
while (timeint(lastInt)>endTime+etol)
    lastInt=lastInt-1;
end
firstepsilon = epsilon;
%calculate rhostar
expArrivals=0;
expService=0;
rhostar=0;
expArrivals= arate(lastInt)*(endTime-timeint(lastInt));
expService= srate(lastInt)*(endTime-timeint(lastInt));
if expService>0, rhostar = expArrivals/expService; end
for m=lastInt-1:-1:j+1
    expArrivals= expArrivals + arate(m)*(timeint(m+1)-timeint(m));
    expService= expService + srate(m)*(timeint(m+1)-timeint(m));
    if expArrivals/expService > rhostar
        rhostar = expArrivals/expService;
    end
end
expArrivals=expArrivals + arate(j)*(timeint(j+1)-startTime);
expService=expService + srate(j)*(timeint(j+1)-startTime);
```

```
if expArrivals/expService > rhostar
    rhostar = expArrivals/expService;
end
% simple memory checking
syscheck = memory;
first = syscheck.MaxPossibleArrayBytes/8 - 10000;
maxmem = syscheck.MemAvailableAllArrays/8-1000000;
if 8*first > maxmem, first = maxmem/8; end
if first < (expArrivals+expService)/epsilon
    epsilon=(expArrivals+expService)/first;
end
timeData=zeros(1,ceil((expArrivals+expService)/epsilon));
sizeData=zeros(1,ceil((expArrivals+expService)/epsilon));
if epsilon>firstepsilon
    sprintf('Memory limits require epsilon to change to %d'...
        , epsilon); end
top=initialSize;
time=startTime;
fulltime=0;
timeData(1) = time;
sizeData(1) = initialSize;
nextservice=-epsilon/srate(j)*log(rand);
% if the service doesn't finish before the interval
```

```
k=j+1;
while (timeint(k)<endTime-etol)&&(nextservice>timeint(k)-time)
    nextservice=timeint(k)-time+...
        ((timeint(k)-time)/nextservice-1)*epsilon/srate(k)...
        *log(rand);
    k=k+1;
end
nextarrival=-epsilon/arate(j)*log(rand);
i=2;
%the simulation runs until the maximum time specified
while (time < endTime-etol)
    %case: the current service finishes before the next arrival
    if nextservice < nextarrival
        %if no customers are in the queue, no service can happen
        if sizeData(i-1)<1-etol
            %so instead, calculate the arrival
            time=time+nextarrival;
            timeData(i)=time;
            sizeData(i)=1;
            nextarrival=-epsilon/arate(j)*log(rand);
            %calculate a new service time
            nextservice=-epsilon/srate(j)*log(rand);
            % if the service doesn't finish before the interval,
```

```
            % let the server run at the rate of the interval, then
            % recalculate the rate for any after that
            k=j+1;
            while (timeint(k)< endTime-etol) &&...
                    (nextservice > timeint(k)-time)
                nextservice = timeint(k)-time+...
                    ((timeint(k)-time)/nextservice-1)...
                    *epsilon/srate(k)*log(rand);
                k=k+1;
            end
        else
%calculate the service
            %increase the time
            time=time+nextservice;
            %set the data to be (time serviced finished,queue-1)
            timeData(i)=time;
            sizeData(i)=sizeData(i-1)-1;
            %recalculate the next arrival and service times
            %the arrival time is the difference between what it
            %last was and the time it took the current service
            nextarrival=nextarrival-nextservice;
            %the next service time is exponentially distributed
            nextservice=-epsilon/srate(j)*log(rand);
```

```
        % if the service doesn't finish before the interval,
        % let the server run at the rate of the interval, then
        % recalculate the rate for any after that
        k=j+1;
        while (timeint(k)< endTime-etol) &&...
                (nextservice > timeint(k)-time)
            nextservice = timeint(k)-time+...
                ((timeint(k)-time)/nextservice-1)...
                *epsilon/srate(k)*log(rand);
            k=k+1;
        end
    end
%other case: the next arrival happens before the current
%service finishes
else
    %set the time
    time=time+nextarrival;
    % calculate empty time
    timeData(i)=time;
    sizeData(i)=sizeData(i-1)+1;
    %find the top of the graph
    if sizeData(i)>top+etol, top=top+1; end
    %recalculate the next service and arrival times
```

```
        nextservice=nextservice-nextarrival;
        nextarrival=-epsilon/arate(j)*log(rand);
    end
    %move to the next interval if necessary
    while (timeint(j)< endTime-etol) && (timeint(j+1)<time)
        j=j+1; end
    i=i+1;
end
figure
plot(timeData(1:i-1),sizeData(1:i-1));
axis([startTime endTime 0 top]);
title(['Queue Length with rho* = ' num2str(rhostar,'%1.4f')...
    ', epsilon = ' num2str(epsilon,'%1.3f')],'FontSize',12)
xlabel('Time - t')
ylabel('# of Requests in Server - X(t)')
```

```
% queuingsystemcalc.m
% Allen Flick Master's Thesis Queuing System Simulation
% Simulates a Queuing system and calculates probabilities
% Customers arrive according to a non-homogeneous poisson process
% of rate 'arate', and are served with rate 'srate', both of which
% are piecewise constant functions of time.  If no customers
% are present, the first customer's service begins immediately
% when he arrives.
function queuingsystemcalc(initialSize, arate, srate, timeint,...
    startTime, endTime, epsilon, runTimes, outputFile)
%default values
if nargin < 1
    arate=[7,3];
    srate=[3,9];
    timeint=[0,2,50];
    startTime=timeint(1);
    endTime=timeint(end);
    initialSize=2;
    epsilon=0.00001;
    runTimes=200000;
    outputFile='comparison.txt';
end
%initialize variables
```

```
etol=.0000001;
cleartime=0;
j=1;
while (timeint(j)<startTime-etol)
    j=j+1;
end
if timeint(j)-startTime>etol
    j=j-1;
end
firstInt=j;
lastop=0;
firstepsilon = epsilon+0;
lastInt=length(timeint)-1;
while (timeint(lastInt)>endTime+etol)
    lastInt=lastInt-1;
end
%calculate rho*
expArrivals=0;
expService=0;
rhostar=0;
expArrivals= arate(lastInt)*(endTime-timeint(lastInt));
expService= srate(lastInt)*(endTime-timeint(lastInt));
if expService>0
```

```
    rhostar = expArrivals/expService;
end
for m=lastInt-1:-1:j+1
    expArrivals=expArrivals+arate(m)*(timeint(m+1)-timeint(m));
    expService=expService+srate(m)*(timeint(m+1)-timeint(m));
    if expArrivals/expService > rhostar
        rhostar = expArrivals/expService;
    end
end
expArrivals=expArrivals + arate(j)*(timeint(j+1)-startTime)
expService=expService + srate(j)*(timeint(j+1)-startTime)
if expArrivals/expService > rhostar
    rhostar = expArrivals/expService;
end
% simple memory checking
syscheck = memory;
first = syscheck.MaxPossibleArrayBytes/8 - 10000;
maxmem = syscheck.MemAvailableAllArrays/8-1000000;
if 3*first > maxmem
    first = maxmem/3;
end
if first < expArrivals/epsilon
    epsilon=expArrivals/first;
```

```
end
if epsilon>firstepsilon
    sprintf('Memory limits require epsilon to change to %d'...
        , epsilon)
end
sizeTime=zeros(1,ceil(expArrivals/epsilon));
runFrac=zeros(1,ceil(max(100,initialSize+expArrivals/.01)));
time=startTime;
top=initialSize;
queueSize=initialSize;
nextservice=-epsilon/srate(j)*log(rand);
% if the service doesn't finish before the interval
k=j+1;
while (timeint(k)<endTime-etol)&&(nextservice>timeint(k)-time)
    nextservice = timeint(k)-time...
        +((timeint(k)-time)/nextservice-1)...
        *epsilon/srate(k)*log(rand);
    k=k+1;
end
nextarrival=-epsilon/arate(j)*log(rand);
i=2;
%run the simulation on all but the last interval
while (time < timeint(lastInt)-etol)
```

```
%case: the current service finishes before the next arrival
if nextservice < nextarrival
    %if no customers are in the queue, no service can happen
    if queueSize<1-etol
        %so instead, calculate the arrival
        time=time+nextarrival;
        queueSize=1;
        nextarrival=-epsilon/arate(j)*log(rand);
        % calculate a new service time
        nextservice=-epsilon/srate(j)*log(rand);
        % if the service doesn't finish before the interval,
        % let the server run at the rate of the interval, then
        % recalculate the rate for any after that
        % be careful with the service jumping intervals
        k=j+1;
        while (timeint(k)< endTime-etol) &&...
                (nextservice > timeint(k)-time)
            nextservice = timeint(k)-time+...
                ((timeint(k)-time)/nextservice -1)...
                *epsilon/srate(k)*log(rand);
            k=k+1;
        end
    else
```

```
% calculate the service
            %increase the time
            time=time+nextservice;
            queueSize=queueSize-1;
            %recalculate the next arrival and service times
            %the arrival time is the difference between what it
            %last was and the time it took the current service
            nextarrival=nextarrival-nextservice;
            %the next service time is exponentially distributed
            nextservice=-epsilon/srate(j)*log(rand);
            % if the service doesn't finish before the interval,
            % let the server run at the rate of the interval, then
            % recalculate the rate for any after that
            % be careful with the service jumping intervals
            k=j+1;
            while (timeint(k)< endTime-etol) &&...
                    (nextservice > timeint(k)-time)
                nextservice = timeint(k)-time+...
                    ((timeint(k)-time)/nextservice-1)...
                    *epsilon/srate(k)*log(rand);
                k=k+1;
            end
        end
```

```
    %other case: the next arrival happens before the current
    %service finishes
    else
        %set the time
        time=time+nextarrival;
        queueSize=queueSize+1;
        %find the max queue size
        if queueSize>top+etol
            top=top+1;
        end
        %recalculate the next service and arrival times
        nextservice=nextservice-nextarrival;
        nextarrival=-epsilon/arate(j)*log(rand);
    end
    %move to the next interval if necessary
    while (timeint(j)< endTime-etol) && (timeint(j+1)<time)
        j=j+1;
    end
    i=i+1;
end
j=lastInt;
%calculate the last interval, until the queue is empty
%the simulation runs until the maximum time specified
```

```
while (time < endTime-etol)
    if nextservice < nextarrival
        if queueSize<1-etol
            time=time+nextarrival;
            queueSize=1;
            nextarrival=-epsilon/arate(j)*log(rand);
            nextservice=-epsilon/srate(j)*log(rand);
            cleartime=time;
            break
        else
            time=time+nextservice;
            queueSize=queueSize-1;
            nextarrival=nextarrival-nextservice;
            nextservice=-epsilon/srate(j)*log(rand);
        end
    else
        time=time+nextarrival;
        queueSize=queueSize+1;
        if queueSize>top+etol
            top=top+1;
        end
        nextservice=nextservice-nextarrival;
        nextarrival=-epsilon/arate(j)*log(rand);
```

```
    end
    i=i+1;
end
%now calculate the rest of the last interval, where we get our
%data for the probabilities
%the simulation runs until the maximum time specified
while (time < endTime-etol)
    if nextservice < nextarrival
        if queueSize<1-etol
            time=time+nextarrival;
            sizeTime(1)=sizeTime(1)+nextarrival;
            queueSize=1;
            nextarrival=-epsilon/arate(j)*log(rand);
            nextservice=-epsilon/srate(j)*log(rand);
        else
            time=time+nextservice;
            sizeTime(queueSize+1)=sizeTime(queueSize+1)...
                +nextservice;
            queueSize=queueSize-1;
            nextarrival=nextarrival-nextservice;
            nextservice=-epsilon/srate(j)*log(rand);
        end
    else
```

```
        time=time+nextarrival;
        sizeTime(queueSize+1)=sizeTime(queueSize+1)+nextarrival;
        queueSize=queueSize+1;
        if queueSize>top+etol
            top=top+1;
        end
        nextservice=nextservice-nextarrival;
        nextarrival=-epsilon/arate(j)*log(rand);
    end
    i=i+1;
end
%calculate the long run fractions of time in each state
totaltime=0;
for m=1:length(sizeTime)
    totaltime=totaltime+sizeTime(m);
end
for m=1:length(sizeTime)
    sizeTime(m)=sizeTime(m)/totaltime;
end
% run a number of non-accelerated simulations for comparison
for trial=1:runTimes
j=firstInt;
time=startTime;
```

```
queueSize=initialSize;
nextservice=-1/srate(j)*log(rand);
% if the service doesn't finish before the interval
k=j+1;
while (timeint(k)<endTime-etol)&&(nextservice>timeint(k)-time)
    nextservice = timeint(k)-time...
        +((timeint(k)-time)/nextservice -1)...
        *1/srate(k)*log(rand);
    k=k+1;
end
nextarrival=-1/arate(j)*log(rand);
i=2;
%run the simulation without epsilon acceleration
while (time < endTime-etol)
    %case: the current service finishes before the next arrival
    if nextservice < nextarrival
        %if no customers are in the queue, no service can happen
        if queueSize<1-etol
            %so instead, calculate the arrival
            time=time+nextarrival;
            queueSize=1;
            lastop=-1;
            nextarrival=-1/arate(j)*log(rand);
```

```
        % calculate a new service time
        nextservice=-1/srate(j)*log(rand);
        % if the service doesn't finish before the interval,
        % let the server run at the rate of the interval, then
        % recalculate the rate for any after that
        % be careful with the service jumping intervals
        k=j+1;
        while (timeint(k)< endTime-etol) &&...
                (nextservice > timeint(k)-time)
            nextservice = timeint(k)-time+...
                ((timeint(k)-time)/nextservice-1)...
                *1/srate(k)*log(rand);
            k=k+1;
        end
    else % calculate the service
        %increase the time
        time=time+nextservice;
        queueSize=queueSize-1;
        lastop=1;
        %recalculate the next arrival and service times
        %the arrival time is the difference between what it
        %last was and the time it took the current service
        nextarrival=nextarrival-nextservice;
```

```
        %the next service time is exponentially distributed
        nextservice=-1/srate(j)*log(rand);
        % if the service doesn't finish before the interval,
        % let the server run at the rate of the interval, then
        % recalculate the rate for any after that
        % be careful with the service jumping intervals
        k=j+1;
        while (timeint(k)< endTime-etol) &&...
                (nextservice > timeint(k)-time)
            nextservice = timeint(k)-time+...
                ((timeint(k)-time)/nextservice-1)...
                *1/srate(k)*log(rand);
            k=k+1;
        end
    end
%other case: the next arrival happens before the current
%service finishes
else
    %set the time
    time=time+nextarrival;
    queueSize=queueSize+1;
    lastop=-1;
    %recalculate the next service and arrival times
```

```
        nextservice=nextservice-nextarrival;
        nextarrival=-1/arate(j)*log(rand);
        %maybe use lack of memory here
    end
    %move to the next interval if necessary
    while (timeint(j)< endTime-etol) && (timeint(j+1)<time)
        j=j+1;
    end
    i=i+1;
end
runFrac(queueSize+1+lastop)=runFrac(queueSize+1+lastop)+1;
end %end for runTimes section
%calculate the ratio of times in each state.
for m=1:length(runFrac)
    runFrac(m)=runFrac(m)/runTimes;
end
% write data to a file
fid = fopen(outputFile, 'wt');
fprintf(fid,'\t   With epsilon = %1.7f, ',epsilon);
fprintf(fid,'the system cleared out\n');
fprintf(fid,'\t    at %2.5f after having ',cleartime);
fprintf(fid,'a maximum queue length\n');
fprintf(fid,'\t    of %d.   Ran %d times ',top,runTimes);
```

```
fprintf(fid,'without acceleration.\n\n');
fprintf(fid,'\tCalculated Long Run Fractions ');
fprintf(fid,'of Time / Probabilities\n');
fprintf(fid,'\tThat is, P[X = n] for each state below.\n');
fprintf(fid,'\tn\tTheoretical\t  Accelerated\tMultiple Runs');
for m=1:20%min(100,length(runFrac))
    fprintf(fid,'\n\t%d\t%1.8f\t  %1.8f\t%1.8f',m-1,...
        (arate(lastInt)/srate(lastInt))^(m-1)...
        *(1-arate(lastInt)/srate(lastInt)),...
        sizeTime(m),runFrac(m));
end
fclose(fid)
```