INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

Université d'Ottawa · University of Ottawa

Leaky Bucket Analysis
by Means of Separate Mutually Dependent Queues

By

Bassem Hajjar

A thesis submitted to the
School of Graduate Studies and Research
in partial fulfillment of the requirements of the degree of

Masters of Applied Sciences

Ottawa-Carleton Institute for Electrical Engineering
School of Information Technology and Engineering (SITE)
University of Ottawa

June 1999

Abstract

The leaky bucket admission control scheme has been introduced as the congestion control mechanism of choice for Broadband networks. The leaky bucket is flexible and capable of accommodating fluctuations in traffic. In today's high-speed networks, it is preferred over traditional window based flow control as a traffic regulation and congestion avoidance mechanism.

This thesis attempts to model the performance of the leaky bucket mechanism by analyzing it in two separate yet mutually dependent queues. This method, due to its simplicity, can easily model variations of the leaky bucket scheme. By separating the analysis of the two queues, the model can be quickly adapted if the assumptions change. In addition, the thesis uses the results to analyze the performance of two variants of the leaky bucket.

Acknowledgements

I would like to express my sincere gratitude to Dr. Oliver W. Yang, my research supervisor, for his help and guidance throughout this research.

I also wish to extend my thanks to my managers at Nortel Networks, Gordon Blair, Jim Davidson and Cuong Duong, and also to my colleagues (both at Nortel Networks and in the Computer Communication Networks Group) and my parents for their support and understanding.

Table of Contents

Abstract ____ *i*

Acknowledgements ____ *ii*

Table of Contents ____ *iii*

List of Figures ____ *v*

Acronyms, Notations an Symbols ____ *vi*

Chapter 1 Introduction ____ *1*

1.1 Overview ____ 1

1.2 Literature Review ____ 3

1.3 Motivation and Approaches ____ 6

1.4 Objectives and Contributions ____ 8

1.5 Thesis Organization ____ 8

Chapter 2 Modeling and Assumptions ____ *10*

2.1 The Leaky Bucket Model ____ 10

2.2 Assumptions ____ 10

Chapter 3 Analysis ____ *12*

3.1 Leaky Bucket System Analysis ____ 12

3.1.1 Packet Buffer Analysis ____ 12

3.1.2 Token Pool Analysis ____ 13

3.1.3 Packet Buffer Length Equation ____ 14

3.1.4 D/M/1/*K* Queue Analysis ____ 15

3.2 Analysing Variations of the Leaky Bucket ____ 16

3.2.1 Leaky Bucket with Finite Storage ____ 17

3.2.2 Leaky Bucket with Bulk Arrival ____ 19

3.2.3 Packet Buffer Analysis with an Exceptional First Service ____ 20

Chapter 4 Validation ____ *23*

4.1 Mean Queue Length Analysis ____ 23

4.2 D/M/1/*K* Computations ____ 26

4.3 Simulation ____ 27

4.4 Results ____ 30

4.5 Discussion of the results ____ 31

Chapter 5 Performance Evaluation ____ *34*

5.1 Mean Queue Length ____ 34

5.2 Mean Waiting Time 36
5.3 Packet Loss Probability 37
Chapter 6 Engineering Applications 42
6.1 Design Guidelines 42
6.2 Application Example for the Mean Queue Length and Waiting Time 44
Chapter 7 Conclusion 45
7.1 Future work 45
References 47
Appendix I: Maple V Code 49
Appendix II: OPNET 4.0 Simulation Model 56

List of Figures

Figure 1 Model of the leaky bucket system 2

Figure 2 State transition diagram of the model used in [3] 4

Figure 3 Leaky Bucket with finite storage model 17

Figure 4 Packet arrival to a buffer made empty by a token arrival 24

Figure 5 Packet arrival to a buffer made empty by a packet arrival 25

Figure 6 Node diagram of the OPNET leaky bucket model 28

Figure 7 Simulation and Analysis Results for Token Pool Size, B, of 6 and 12. 31

Figure 8 Example of the effect of the embedded Markov points on the results of the D/M/1/K analysis 33

Figure 9 Mean queue length vs. arrival rate for different values of B. (Linear scale) 34

Figure 10 Mean queue length vs. arrival rate for different values of B. (Logarithmic scale) 35

Figure 11 Mean queue length vs. token pool size for different values of ρ. 36

Figure 12 Normalized mean waiting time as seen by arriving packets. 37

Figure 13 Packet loss probability for B = 6 and different values of K. 38

Figure 14 Packet loss probability for K = 6 and different values of B. 39

Figure 15 Packet loss probability vs. packet buffer size, K, for B = 6. 40

Figure 16 Packet loss probability vs. token pool size, B, for K = 6. 41

Acronyms, Notations an Symbols

This section lists the acronyms, notations and symbols used throughout the thesis.

Table 1: ***Acronyms***

Acronym	Description	First occurrence on page
D/M/1	Queue with a Deterministic arrival, Markovian service process and 1 server.	4
M/G/1	Queue with a Markovian arrival process, General service process and 1 server.	7
D/M/1/*K*	Queue with a Deterministic arrival process, Markovian service process, one server and a waiting room of size *K*.	8
$\mathfrak{L}$	Laplace Transform.	13
G/M/1	Queue with a General arrival process, Markovian service process and 1 server.	15
pdf	Probability density function.	13
OPNET	OPtimized Network Engineering Tools.	27
FIFO	First In First Out.	10

Table 2: ***Notations***

Notation	Description	First occurrence on page
Italic typeface	Denotes a variable.	10
Bold/italic typeface	Denotes a matrix or vector variable.	16
Pr[x]	Probability that x occurs.	12
Pr[x|y]	Probability that x occurs conditioned on y.	12
Q(z|y)	Q(z) conditioned on y.	12
E[x]	Expectation (or mean) of x.	24
$\frac{\partial Q(z)}{\partial z}$	Derivative of $Q(z)$ with respect to z.	26

$Q^{(n)}(z)$	The n^{th} derivative of $Q(z)$ with respect to z.	18

Table 3: Symbols

Symbol	Description	First occurrence on page
B	Size of the token pool.	10
λ	Packet arrival rate.	10
λ_T	Token arrival rate.	10
δ	Token interarrival time. $\delta = 1/\lambda_T$	10
q	Number of packets in the packet buffer.	12
q_T	Number of tokens in the token pool.	12
$Q(z)$	The z-transform of the packet buffer length.	12
$Q_T(z)$	The z-transform of the token pool length.	12
ρ	Normalized packet arrival rate. $\rho = \lambda/\lambda_T$	13
$\mathfrak{B}(s)$	The Laplace Transform ($\mathfrak{LS}$) of the pdf of the token interarrival with an initial residual time.	13
$\Theta(z)$	The z-transform of the queue length in a D/M/1/K system.	13
μ	Service rate of the D/M/1/K queue.	15
K	Size of the D/M/1/K queue.	15
δ	Interarrival time to the D/M/1/K queue (same as the token arrival).	15
X_i	The number of customers in a D/M/1/K queue that the i^{th} arrival sees upon joining the queue.	15
$\mathfrak{B}_n$	The number of customers served between the n^{th} and the $(n+1)^{st}$ arrivals to the D/M/1/K queue.	15
p_{ij}	The probability of having i customers in the D/M/1/K queue at the n^{th} arrival and j customers at the $(n+1)^{st}$ arrival.	15
b_n	The probability that n customers are served during an interarrival time in a D/M/1/K queue	16
π_x	Probability of having x customers in the D/M/1/K queue.	16
$\boldsymbol{\pi}$	Probability vector of the number of customers in a D/M/1/K queue. $\boldsymbol{\pi} = [\pi_0, \pi_1, \pi_2, \ldots, \pi_K]$	16
$\boldsymbol{P}$	Transition probability matrix of the D/M/1/K.	16
$\boldsymbol{I}$	The Identity matrix.	26
$\boldsymbol{0}$	The all zero vector.	26
$\Phi(z)$	The z-transform of the queue length in a D/M/1/K queue.	17
$q(K)$	The probability that the packet buffer contains K packets.	18

$\Phi^{(n)}(z)$	The n^{th} derivative of $\Phi(z)$ with respect to z.	18
$q^*(x)$	The probability of having x packets in an M/G/1/K queue.	18
$G_B(z)$	The probability generating function of the batch size in a system with bulk arrival.	19
m_B	The mean batch size in a system with bulk arrival.	19
$\tilde{\mathcal{B}}(s)$	The Laplace Transform of the service time pdf, which is the token interarrival time pdf.	21
$\tilde{\mathcal{F}}(s)$	The Laplace Transform of the first (exceptional) service time pdf.	21
m_F	The mean of the first service time.	21

Chapter 1

Introduction

The advent of high-speed networks brought with it new challenges in the areas of congestion control. These challenges are introduced due to the nature of the networks where the propagation delays became the dominant factor rather than queueing or transmission delays. Many of the techniques used for traditional lower speed networks could no longer be relied upon to provide the quality of service required by the emerging applications.

Two main techniques are normally considered for congestion control: reactive and preventive. In reactive control, the congestion is detected after it occurs and appropriate measures are taken in reaction. Preventive congestion control, however, tries to prevent congestion before it occurs [10]. This technique is normally implemented by shaping the incoming traffic to somehow give it more desirable characteristics. Perhaps the most popular preventive congestion control mechanism for high-speed networks is the leaky bucket.

1.1 Overview

The leaky bucket was proposed by J. Turner in 1986 [4] as a traffic policing and shaping mechanism for data at the access point of the network. This mechanism is based on the use of tokens, generated at a regular rate, to give packets access to the network at a certain maximum rate, which is the rate for token generations. Although this mechanism resembles, literally, a leaky bucket, the "leaky bucket" term is currently used for a more general scheme that was proposed later. This mechanism is described below:

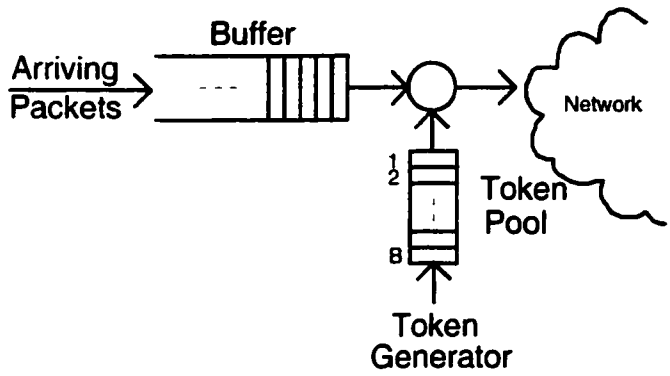


Figure 1 *Model of the leaky bucket system*

The system, as depicted in Figure 1, consists of a token generator, a token pool and an optional packet buffer. Packets arrive at the packet buffer and get queued. For a packet in the buffer to leave the controller and be admitted into the network, it must obtain a token. Tokens are generated at a constant rate λ_T and arrive at the token queue (or token pool). The token pool has a finite size B, i.e. if there are B tokens already in the token pool, any newly generated tokens are discarded. If a packet arriving in the packet buffer finds no tokens in the pool it must wait for the next token to be generated. Alternatively, in a system with no packet buffer, it may be discarded or admitted to the network but marked as discardable, should congestion be encountered (as the connection has violated its negotiated throughput). When a packet leaves, it removes one token from the pool.

The purpose of the leaky bucket is:

1. The long-run average rate at which packets can leave the system is upper bounded by λ_T.

2. If a very large number of packets arrive back-to-back in a long burst, at most B of these packets can leave back-to-back; after that, the remaining packets will depart with a separation of $1/\lambda_T$ between successive packets.

So, regardless of the input, traffic on the output side of the leaky bucket would alternate between periods in which relatively short bursts of packets are clustered together and

periods in which strings of packets have a minimum separation of $1/\lambda_T$. Clearly, this will smooth out the traffic at the cost of some transformation of the input traffic and delay [2].

The leaky bucket is analyzed in order to determine the effect of the parameters, such as the arrival rates, the packet buffer size and the token pool size, on the performance of the system. The analysis also helps in selecting the proper parameters to guarantee a quality of service (QoS) in terms of throughput, delay and packet loss for instance.

1.2 Literature Review

This section surveys a number of papers where the leaky bucket is studied analytically. The main emphasis of this review is pointing out the assumptions and methods used in analyzing the leaky bucket system.

In [6] an ATM network environment is assumed where only fixed size packets (or cells) are transmitted over the network. Since cell arrivals often correspond to the segmentation of large packets into many cells or the superposition of a number of cells coming from different logical connections over the same path in the network, the arrival patterns are represented with a batch arrival process. Due to the fixed packet size a discrete-time environment, where the time is slotted, is considered. The distribution of batch sizes is arbitrary while the time between arrivals of successive batches is taken to be geometrically distributed. The state of the leaky bucket system is described at every time slot for the number of cells in the buffer, the number of tokens in the pool, and the phase of the token generation process (since it takes multiple time slots for a token to be generated). The result is a three-dimensional Markov chain. From the state equations a transition probability matrix for the Markov chain is obtained. Matrix Analytic techniques are then applied to obtain the queue length distribution with respect to the parameters of the leaky bucket system.

Reference [5] uses worst case analysis to study the effects of the leaky bucket parameters on the average queueing delay. The worst case traffic is characterized as the repetition of

the following three phases: bulky arrivals, arrival at every token generation for a specified length of interval, and then no arrival till the token bucket is full.

Rather than using a discrete time Markov chain approach, which is used in most papers to find the steady state of the system, the time-dependent behavior of the leaky bucket was studied in [3]. The model is based on the theory of Markov regenerative processes and is studied for Poisson and ON-OFF sources. The ON-OFF arrival process is supposed to model the bursty characteristics of ATM traffic. The state representation of the system is defined with a pair – the number of packets and the number of tokens in the system at one time. The token and packet transmission in this model is assumed to be instantaneous, therefore, the token pool and the packet buffer cannot be non-empty at the same time. The state transition diagram then reduces to one similar to a D/M/1 system with a queue size equal to the combined sizes of the packet buffer and the token pool as shown in Figure 2. The only real difference from the regular D/M/1 queue is that the deterministic service does not stop when there are no customers in the queue.

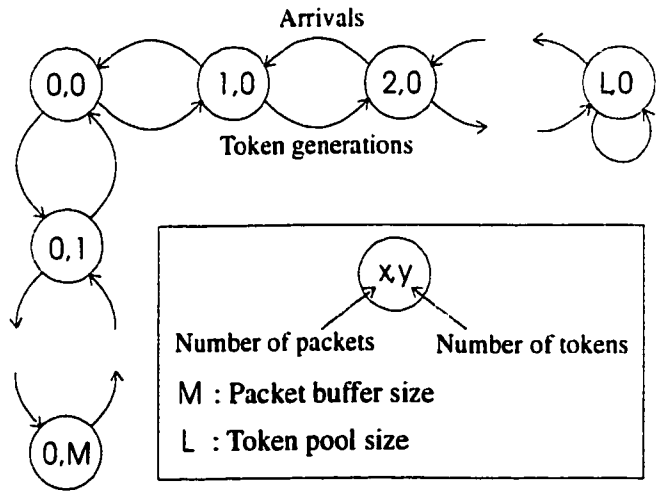


Figure 2 *State transition diagram of the model used in [3]*

The combined queue concept is also used in [7]. The model assumes a Poisson packet arrival, deterministic token arrival and a zero service time. The system is analyzed, at token generation instants, for the number of packets and tokens in the packet and token

queues respectively. The zero service time assumption reduces the two-dimensional state description to one dimension since the packets queue if and only if there are no tokens in the token pool. The paper finds the packet waiting time distribution and the Laplace transform of the interdeparture time distribution. The Laplace transform is used to find the squared coefficient of variation of the interdeparture time. The results are used to find a tradeoff between the smoothing effect of the leaky bucket on the departing traffic and the packet waiting time.

A discrete time analysis of the leaky bucket is performed in [8]. The model considers the following assumptions: (i) The data arrivals follow a discrete time Poisson process in one case, and a Markov Modulated Poisson Process (MMPP) in the other; (ii) The token generating intervals are constant; (iii) The token pool is finite while the data buffer can be finite or infinite; (iv) The data units are assumed to be cells of a fixed length which gives them a deterministic service time. The analysis assumes that the discrete time slots are long enough to transmit one cell and that the token interarrival time is a multiple of the slot time. The system state is described at each time slot and any packet or token arrivals are considered only at the end of the time slot. It becomes clear, in this case, that either the packet buffer or the token pool must be empty after a possible transmission at the beginning of a slot. This is because, when the system state is described at the end of the time slot, all transmissions would have been completed until either the packet buffer or the token pool are depleted. This assumption simplifies the analysis considerably. It is equivalent to assuming a zero service time. A further assumption, of a short slot time compared to the mean interarrival time, leads to approximating the Poisson arrival by a Bernoulli process. The fact that at least one of the queues must be empty, at each point the system is described, helps to reduce the dimension of the state transitions by defining the state of the system as the difference between the number of queued tokens and the number of buffered packets. This difference is called the deficit function.

The deficit function, which is somehow similar to combining the two queues, is also used in [9]. Due to the instantaneous transmission of packets and tokens, the state of the system is described by $z(t) = x(t) - y(t)$ where $x(t)$ and $y(t)$ are the number of packets and

tokens at time t respectively. To avoid the negative values sometimes the token pool size is added to the equation. This is the case in [12] and also in [2], where a conformance measure of the leaky bucket output traffic to the input traffic is found.

The instantaneous transmission of packets (or zero service time) is always a recurring assumption as seen again in [14]. Whenever this assumption is not present a discrete-time analysis is performed with assumptions similar to [8]. Such is the case of [13] where, in addition to the usual steady-state distribution of the queue length and the waiting time, the probability generating function of the interdeparture times is found. The deficit function is again used in this paper.

A Markov Modulated Bernoulli Process (MMBP) is assumed for the packet arrivals in [10]. The immediate transmission of packets (or cells) is assumed in this paper – as in the others. This assumption also helps in reducing the states of the system by allowing the use of the deficit function described above since both queues cannot be non-empty at the same time. The analysis performed is based on a Markov chain embedded at the token arrival instants. The paper develops expressions for the cell loss probability of cells entering the leaky bucket and for the mean interdeparture time of cells leaving the leaky bucket. It also develops approximations for the squared coefficient of variation of the interdeparture time.

Finally, a new class of traffic – the long-range dependent – is studied at the input of the leaky bucket in [11]. The paper assumes that the cell arrival process belongs to a class of discrete-time long-range dependent traffic models. The analysis relies on time slots at the beginning of which the arrivals, both packets and tokens, are considered. The packet transmission is again considered instantaneous as in all the previous references.

1.3 Motivation and Approaches

Performance analysis of the leaky bucket using queueing theory has usually been carried out using certain assumptions. These assumptions are necessary to simplify the analysis,

however, they tend to limit the application of the approach. The methods used cannot easily be modified to describe variations of the system – like a non deterministic token arrival or a non zero service time – since these assumptions are essential for the method to work. All of the papers discussed in the previous section considered either an instantaneous packet transmission (i.e. a zero service time), or a fixed transmission time but with a discrete-time approach where the transmission is completed before the next time slot.

My first attempt at analyzing a leaky bucket with a general service time was carried out using "M/G/1 with Vacation" queueing results from [15]. The idea was to model the leaky bucket's packet buffer as an M/G/1 queue where the absence of tokens forces the server to go on "vacation" and cease servicing the packets. This meant that the server and, hence, the packet buffer were dependant on the state of the token pool, which was not difficult to model. The main difficulty, however, was in trying to describe the behavior of the token pool since it depended, in its turn, on the state of the packet buffer. This mutual dependence turned out to be the major obstacle that rendered the attempt unsuccessful since none of the results in [15] could model vacations that depended on the state of the M/G/1 queue.

While searching for a solution to the issue of mutual dependence between the two queues, this thesis was started with the idea of using queueing theory to analyze each queue separately based on certain states of the other queue and then combining them to form a single equation describing the complete system.

Although the work done in this thesis does not consider a non zero service time, which was the main motivation behind our original "M/G/1 with vacations" attempt, but it proves the viability of the mutually dependent queues analysis approach in describing the leaky bucket system.

1.4 Objectives and Contributions

The objective of this thesis is to prove that the leaky bucket system can be modeled by analyzing the packet and token queues as separate yet mutually dependent queues. This is achieved by finding a standalone equation describing the behavior of the packet buffer. To validate the result an approximation of the mean queue length is found and a simulation model is designed in order to compare the outcome of both methods.

Since the main contribution of this thesis is to demonstrate the validity of the followed approach, two variants of the leaky bucket are analyzed, namely the finite storage and the bulk arrival systems. The results obtained for the approximate mean queue length and the mean waiting time of the infinite storage system and of the loss probability of the finite storage system are plotted. These plots are used to analyze the performance of these systems adding their application in engineering problems to the scope of the thesis.

Finally, the thesis presents the code written to compute the performance for different values of the system parameters. It also lists the complete model developed to simulate the leaky bucket system.

An additional contribution of this thesis is the analysis of the D/M/1/K queue.

1.5 Thesis Organization

In the first section of the analysis, the leaky bucket queues (the packet buffer and the token pool) are mathematically analyzed separately using queuing theory methods. As the queues are mutually dependent, their equations are then combined together in a single equation summarizing the behavior of the packet buffer in the system.

Due to the unavailability of results for the D/M/1/K queue, required to analyze the token pool, the D/M/1/K queue is analyzed, in the section that follows, using an embedded Markov chain model.

The analysis is then taken further by studying the finite buffer and the bulk arrival leaky buckets.

In the second part of the thesis the resulting equation is validated. First, the mean queue length is determined. With the use of an approximation, the mean queue length is computed for varying system parameters. A simulation model is then introduced. The results of which are compared against the analytical approximation in order to prove the validity of the analysis.

In the performance evaluation section, the results of the analyzed systems are plotted against different system parameters to study the performance.

In the last section some guidelines are presented to help in using the results in engineering applications.

Chapter 2

Modeling and Assumptions

2.1 The Leaky Bucket Model

The leaky bucket queueing model used in this analysis consists of a server with two queues (see Figure 1). The first queue is the packet buffer. It is a First In First Out (FIFO) queue with unlimited size. The token pool is also a FIFO queue but with a limited size B. When a packet arrives and finds a token available, the server admits it to the network and removes the token from the token pool. If there are no tokens the packet is queued in the packet buffer. As soon as a token arrives the first packet in the packet buffer is admitted to the network. If there are no packets available the token is queued in the token pool.

2.2 Assumptions

The following assumptions are used in the analysis:

1. The arrival of packets to the network is Poisson with rate λ.

2. The token generation is deterministic (which is usually the case in real networks) with rate λ_T and period $\delta = 1/\lambda_T$.

3. The service time is zero. It is assumed that the service of the packets, which is purely forwarding, cannot be slower than receiving the packet and at the same time it is fast compared to the token interarrival time. Therefore, a packet will not wait in the buffer while another packet is being transmitted. The zero service time introduces a great simplification to the analysis since it rules out the possibility of having packets and tokens, in the leaky bucket system, at the same time. This is because the packets that find tokens leave the leaky bucket immediately until there are no more packets or no

more tokens, thus leaving at least one of the queues empty. This assumption is consistent with the analysis carried out in all the literature surveyed in Section 1.2.

Chapter 3

Analysis

The difficulty in analyzing the leaky bucket packet buffer lies in its dependence on the token pool. The token pool, in its turn, depends on the behavior of the packet buffer. To analyze the system we look at the token pool and the packet buffer as separate yet mutually dependent queues.

3.1 Leaky Bucket System Analysis

We begin by introducing the notation as follows. Let q be the number of packets in the packet buffer and q_T the number of tokens in the token pool. We also define $Q(z)$ as the z-transform of the packet buffer length, that is

$$Q(z) = \sum_{i=0}^{\infty} \Pr[q = i] z^i$$

and $Q_T(z)$ as the z-transform of the token pool length, that is

$$Q_T(z) = \sum_{i=0}^{B} \Pr[q_T = i] z^i$$

where B is the token pool size.

3.1.1 Packet Buffer Analysis

The first queue to be considered is the packet buffer. This queue must be empty if the token pool is not empty since every arriving packet finds a token and leaves, as explained in Section 2.2. Therefore, the conditional probability of the packet buffer length being equal to zero when the token pool is not empty must be equal to one, i.e. $\Pr[q = 0 | q_T > 0] = 1$. This is expressed in the z-transform domain as

$Q(z|q_T > 0) = 1$.

If the token pool is empty packets start to queue. The packet buffer starts behaving like an M/G/1 queue where the Poisson arrival is the packet arrival at rate λ and the general service time is the deterministic token interarrival time with an initial residual time (the time between the token pool becoming empty and the arrival of the first token). From the M/G/1 results [1]

$$Q(z \mid q_T = 0) = \frac{\mathfrak{B}(\lambda - \lambda z)(1-\rho)(1-z)}{\mathfrak{B}(\lambda - \lambda z) - z}$$

where $\mathfrak{B}(s)$ = Laplace Transform ($\mathfrak{LT}$) of the pdf of the token interarrival with an initial residual time (as described in Section 4.1), and $\rho = \lambda/\lambda_T$.

The two conditional equations are then put together to form the following equation

$$\begin{aligned} Q(z) &= \sum_{i=1}^{B} \Pr[q_T = i] + \Pr[q_T = 0] \frac{\mathfrak{B}(\lambda - \lambda z)(1-\rho)(1-z)}{\mathfrak{B}(\lambda - \lambda z) - z} \\ &= 1 - \Pr[q_T = 0] + \Pr[q_T = 0] \frac{\mathfrak{B}(\lambda - \lambda z)(1-\rho)(1-z)}{\mathfrak{B}(\lambda - \lambda z) - z} \\ &= 1 - Q_T(0) + Q_T(0) \frac{\mathfrak{B}(\lambda - \lambda z)(1-\rho)(1-z)}{\mathfrak{B}(\lambda - \lambda z) - z} \end{aligned}$$

$$Q(z) = 1 - Q_T(0)\left(1 - \frac{\mathfrak{B}(\lambda - \lambda z)(1-\rho)(1-z)}{\mathfrak{B}(\lambda - \lambda z) - z}\right) \qquad (1)$$

3.1.2 Token Pool Analysis

Similarly we analyze the token pool queue. When the packet buffer is non-empty the token pool must be empty. This is described as $\Pr[q_T = 0 | q > 0] = 1$ and the z-transform is $Q_T(z \mid q > 0) = 1$.

When the packet buffer is empty the token pool behaves like a D/M/1/K where K=B, the token pool size. This is because tokens arrive to the pool at the deterministic rate λ_T and depart whenever a packet arrives, which is a Poisson process of rate λ. Therefore,

$Q_T(z \mid q = 0) = \Theta(z)$

where $\Theta(z)$ is the z-transform of the queue length in a D/M/1/K system.

The two conditional equations are put together to give

$$\begin{aligned} Q_T(z) &= \sum_{i=1}^{\infty} \Pr[q=i] + \Pr[q=0]\Theta(z) \\ &= 1 - \Pr[q=0] + \Pr[q=0]\Theta(z) \\ &= 1 - Q(0) + Q(0)\Theta(z) \end{aligned} \tag{2}$$

3.1.3 Packet Buffer Length Equation

The two queues are then combined by putting their mutually dependent equations (1) and (2) together. This yields

$$Q(z) = 1 - (1 - Q(0) + Q(0)\Theta(0))\left(1 - \frac{\mathfrak{B}(\lambda - \lambda z)(1-\rho)(1-z)}{\mathfrak{B}(\lambda - \lambda z) - z}\right) \tag{3}$$

To find $Q(0)$ from equation (3) we evaluate it at $z = 0$ as follows

$$\begin{aligned} Q(0) &= 1 - (1 - Q(0) + Q(0)\Theta(0))\left(1 - \frac{\mathfrak{B}(\lambda)(1-\rho)(1)}{\mathfrak{B}(\lambda)}\right) \\ &= 1 - (1 - Q(0) + Q(0)\Theta(0))(1 - (1-\rho)) \\ &= 1 - (1 - Q(0) + Q(0)\Theta(0))(\rho) \\ &= 1 - \rho + (\rho - \rho\Theta(0))Q(0) \\ &\Rightarrow \\ Q(0) &= \frac{1-\rho}{1-\rho+\rho\Theta(0)} \end{aligned}$$

inserting this equation back into (3) yields

$$Q(z) = 1 - \left(1 - \frac{1-\rho}{1-\rho+\rho\Theta(0)} + \frac{(1-\rho)\Theta(0)}{1-\rho+\rho\Theta(0)}\right)\left(1 - \frac{\mathfrak{B}(\lambda - \lambda z)(1-\rho)(1-z)}{\mathfrak{B}(\lambda - \lambda z) - z}\right) \tag{4}$$

To find $\Theta(0)$, which is the probability of the D/M/1/K queue is empty, we must analyze the D/M/1/K queue.

3.1.4 D/M/1/*K* Queue Analysis

Let us consider a single server queue with an exponential service of rate μ, a deterministic interarrival time δ and a limited waiting room of size K. The addition of the limit K will introduce a slight variation to the method used in [22] to analyze the G/M/1 queue.

We consider the embedded Markov chain where the Markov points are set just prior to an arrival. The system state, X_i, represents the number in the queue that the i^{th} arrival sees upon joining the system. We then have

$$X_{n+1} = X_n + 1 - \mathcal{B}_n \qquad (\mathcal{B}_n \le X_n + 1; X_n \ge 0),$$

where $\mathcal{B}_n$ is the number of customers served during the interarrival time between the n^{th} and the $(n+1)^{st}$ arrivals. Since the service is exponential with mean rate μ and the interarrival time is deterministic with a value of δ we have

$\mathcal{B}_n = \mathcal{B}$ and

$$\Pr[\mathcal{B} = b] = \frac{e^{-\mu\delta}(\mu\delta)^b}{b!}.$$

Let us define p_{ij} as the probability of having j customers in the queue at the $(n+1)^{st}$ arrival given that there were i customers at the n^{th} arrival. The state of the system, X_n, will range from 0 to K, 0 being an empty system and K a full system. We then have

$$p_{ij} = \Pr[X_{n+1} = j \mid X_n = i] = \begin{cases} \Pr[\mathcal{B} = i - j], & i = K, \text{all } j \\ \Pr[\mathcal{B} = i + 1 - j], & 1 \le j \le i + 1 \le K \\ 0, & i + 1 < j \end{cases}$$

$$= \begin{cases} \dfrac{e^{-\mu\delta}(\mu\delta)^{i-j}}{(i-j)!}, & i = K, \text{all } j \\ \dfrac{e^{-\mu\delta}(\mu\delta)^{i+1-j}}{(i+1-j)!}, & 1 \le j \le i+1 \le K \\ 0, & i+1 < j \end{cases} \tag{5}$$

and

$$p_{i0} = 1 - \sum_{j=1}^{i+1} p_{ij}.$$

We will now use the simplified notation

$$
\begin{aligned}
b_n &= \Pr[n \text{ services during an interarrival time}] \\
&= \Pr[\mathcal{S} = n] \\
&= \frac{e^{-\mu\delta}(\mu\delta)^n}{n!}
\end{aligned}
\tag{6}
$$

From (5) and (6) we obtain the following $(K+1)\times(K+1)$ transition probability matrix

$$
P = [p_{ij}] = \begin{bmatrix}
1-b_0 & b_0 & 0 & 0 & \cdots & 0 \\
1-\sum_{k=0}^{1} b_k & b_1 & b_0 & 0 & \cdots & 0 \\
1-\sum_{k=0}^{2} b_k & b_2 & b_1 & b_0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1-\sum_{k=0}^{K-1} b_k & b_{K-1} & b_{K-2} & b_{K-3} & \cdots & b_0 \\
1-\sum_{k=0}^{K-1} b_k & b_{K-1} & b_{K-2} & b_{K-3} & \cdots & b_0
\end{bmatrix}
$$

where i is the row number and j is the column number, both ranging between 0 and K.

We then define the probability vector $\boldsymbol{\pi}$ as $\boldsymbol{\pi} = [\pi_0, \pi_1, \pi_2, \ldots, \pi_K]$ where π_x is the probability of having x customers in the D/M/1/K queue. The steady state probabilities can then be found by solving the set of linear equations given by

$$\boldsymbol{\pi P} = \boldsymbol{\pi}$$

in addition to the normalizing equation

$$\pi_0 + \pi_1 + \pi_2 + \ldots + \pi_K = 1.$$

The z-transform of the D/M/1/K queue, $\Theta(z)$, is defined as $\Theta(z) = \sum_{i=0}^{K} \pi_i z^i$. Hence $\Theta(0)$, the value needed for equation (4), is simply $\Theta(0) = \pi_0$.

3.2 Analysing Variations of the Leaky Bucket

In this section we analyze variation of the leaky bucket model following the same method used in the previous sections. We start, first, by analyzing the finite storage model.

3.2.1 Leaky Bucket with Finite Storage

In actual implementations of the leaky bucket, the packet buffer will be limited in size. This limit to the number of packets waiting for tokens introduces a new measure of the performance of the system – the packet loss rate, which will be found in this section. Packets that do not find a waiting room, when arriving at the leaky bucket with finite storage, are discarded since the source must have exceeded its allowed or negotiated parameters. As a result of discarding packets, the leaky bucket with finite storage enforces hard limits on the arriving traffic in addition to shaping it.

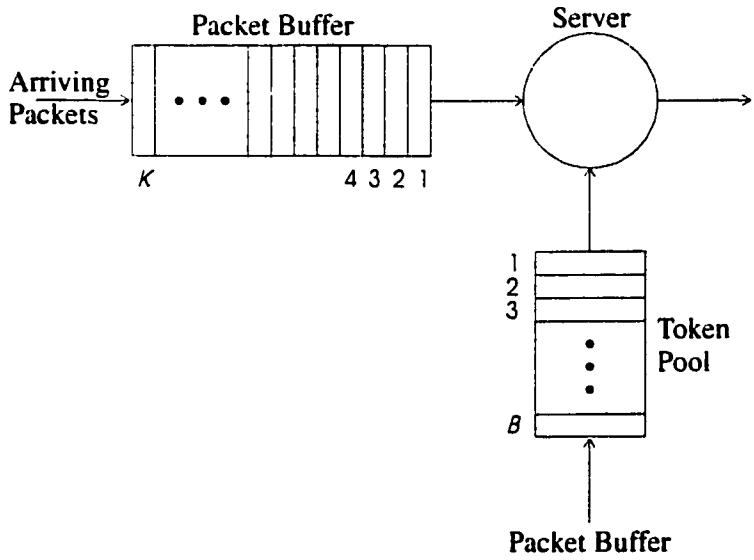


Figure 3 *Leaky Bucket with finite storage model*

The leaky bucket with finite storage model, depicted in Figure 3, is analyzed by simply replacing the M/G/1 results in Section 3.1.1 with the M/G/1/K results. The M/G/1/K model is an M/G/1 queue with a finite storage space of size K. The resulting combined equation, equivalent to equation (3), becomes

$$Q(z) = 1 - (1 - Q(0) + Q(0)\Theta(0))(1 - \Phi(z)) \tag{7}$$

where $\Phi(z)$ is the generating function of the queue length in a D/M/1/K queue.

Setting z to 0, the value of $Q(0)$ is found as

$$Q(0) = \frac{\Phi(0)}{\Phi(0) + \Theta(0) - \Phi(0)\Theta(0)}$$

and equation (7) becomes

$$Q(z) = 1 - \left(1 - \frac{\Phi(0)}{\Phi(0) + \Theta(0) - \Phi(0)\Theta(0)} + \frac{\Phi(0)\Theta(0)}{\Phi(0) + \Theta(0) - \Phi(0)\Theta(0)} \right) (1 - \Phi(z)). \qquad (8)$$

The packet loss rate, that we are trying to find, is equivalent to the probability of a packet arriving to find the packet buffer full. Let $q(K)$ denote the probability that the packet buffer contains K packets. This probability can be found from the probability generating function of equation (8). Since $\Phi(z)$ is the only function of z, equation (8) is rearranged in the form $C_1\, f(z) + C_2$, where C_1 and C_2 are constants and $f(z)$ is a function of z. The rearranged generating function becomes

$$Q(z) = \left(1 - \frac{\Phi(0)}{\Phi(0) + \Theta(0) - \Phi(0)\Theta(0)} + \frac{\Phi(0)\Theta(0)}{\Phi(0) + \Theta(0) - \Phi(0)\Theta(0)} \right) \Phi(z) + \left(\frac{\Phi(0)}{\Phi(0) + \Theta(0) - \Phi(0)\Theta(0)} - \frac{\Phi(0)\Theta(0)}{\Phi(0) + \Theta(0) - \Phi(0)\Theta(0)} \right)$$

and the probability of finding the packet buffer full is

$$q(K) = \frac{Q^{(K)}(0)}{K!} = \left(1 - \frac{\Phi(0)}{\Phi(0) + \Theta(0) - \Phi(0)\Theta(0)} + \frac{\Phi(0)\Theta(0)}{\Phi(0) + \Theta(0) - \Phi(0)\Theta(0)} \right) \frac{\Phi^{(K)}(0)}{K!}$$

$$= \left(1 - \frac{q^*(0)}{q^*(0) + \Theta(0) - q^*(0)\Theta(0)} + \frac{q^*(0)\Theta(0)}{q^*(0) + \Theta(0) - q^*(0)\Theta(0)} \right) q^*(K)$$

where $\Phi^{(K)}(0)$ is the K^{th} derivative of $\Phi(z)$ with respect to z evaluated at $z = 0$, and $q^*(x)$ is the probability of having x packets in an M/G/1/K queue.

The values of $q^*(x)$ are found from the M/G/1/K results in [20] as

$q^*(x) = fp(x)$, for $0 \le x \le K - 1$

and

$$q^*(K) = 1 - \frac{1 - fp(0)}{\rho}$$

where $p(x)$ is the probability of having x packets in an M/G/1 queue with unlimited size and

$$f = \frac{1}{p(0) + \rho \sum_{x=0}^{K-1} p(x)}.$$

From the M/G/1 results we know that

$$p(x) = \frac{\Pi^{(x)}(0)}{x!}$$

where

$$\Pi(z) = \frac{\mathfrak{B}(\lambda - \lambda z)(1-\rho)(1-z)}{\mathfrak{B}(\lambda - \lambda z) - z}$$

with $\mathfrak{B}(s)$ as described in Section 3.1.1.

3.2.2 Leaky Bucket with Bulk Arrival

For performance analysis of high-speed data networks, it is important to consider bulk arrivals to represent the traffic entering the network. The bulkiness of the traffic is due to the fact that cell arrivals correspond, usually, to the segmentation of large customer packets or files. In this section, we re-analyze the leaky bucket using the same method of Sections 3.1.1, 3.1.2, and 3.1.3 except that we assume a batch arrival. Instead of the M/G/1 results used previously we will use the results of the M/G/1 queue with bulk arrivals.

The M/G/1 queue with bulk arrival was analyzed in [21]. The model assumes that customers arrive in batches of random size (given by the generating function $G_B(z)$) and that the batches arrive according to Poisson process of rate λ. The probability generating function of the queue length was found to be

$$\Pi(z) = \frac{(1-\rho)(1-G_B(z))\mathfrak{B}(\lambda - \lambda G_B(z))}{m_B(\mathfrak{B}(\lambda - \lambda G_B(z)) - z)}$$

where $G_B(z)$ is the probability generating function of the batch size, m_B is the mean batch size, and $\mathfrak{B}(s)$ is the Laplace Transform of the service time pdf.

Similar to the work done in the previous section, the results of the pure M/G/1 queue in equation (3) are replaced by those of the M/G/1 queue with bulk arrival – namely the generating function $\Pi(z)$. Equation (3) becomes

$$Q(z) = 1 - (1 - Q(0) + Q(0)\Theta(0))\left(1 - \frac{(1-\rho)(1-G_B(z))\mathcal{B}(\lambda - \lambda G_B(z))}{m_B(\mathcal{B}(\lambda - \lambda G_B(z)) - z)}\right) \quad (9)$$

with $\mathcal{B}(s)$ as described in Section 3.1.1.

Setting z to 0, the value of $Q(0)$ is found as

$$Q(0) = \frac{(1-\rho)(1-G_B(0))}{m_B + (1-\rho)(1-G_B(0))(1-\Theta(0))}$$

and equation (9) becomes

$$Q(z) = 1 - \left(1 - \frac{(1-\rho)(1-G_B(0))}{m_B + (1-\rho)(1-G_B(0))(1-\Theta(0))} + \frac{(1-\rho)(1-G_B(0))\Theta(0)}{m_B + (1-\rho)(1-G_B(0))(1-\Theta(0))}\right)$$
$$\left(1 - \frac{(1-\rho)(1-G_B(z))\mathcal{B}(\lambda - \lambda G_B(z))}{m_B(\mathcal{B}(\lambda - \lambda G_B(z)) - z)}\right)$$

3.2.3 Packet Buffer Analysis with an Exceptional First Service

When we analyzed the packet buffer, we said that it behaves like an M/G/1 queue. Although the service seen by the packets is deterministic (which is the token interarrival time), an M/G/1 queue with a generic service time was considered due to the initial residual service time. The Laplace Transform of the generic service time – $\mathcal{B}(s)$ – is not easy to find due to the different service time that the first packet sees. This is the reason why we will use an approximation in Section 4.1 to find the mean queue length by assuming that the service time is deterministic and that it is equivalent to the mean service time seen by any packet.

In this section we attempt to improve on equation (4) by using M/G/1 with exceptional first service time results instead of the pure M/G/1. This server is ideal in modeling the packet buffer since it allows the separation of the regular service time, which is deterministic, and the initial service time, which will be described in Section 4.1.

The M/G/1 queue with an exceptional first service time is analyzed in [21]. The probability generating function of the queue length is found to be

$$\Pi(z) = \frac{(1-\rho)(\tilde{\mathcal{B}}(\lambda - \lambda z) - z\tilde{\mathcal{F}}(\lambda - \lambda z))}{(1+\lambda m_F - \rho)(\tilde{\mathcal{B}}(\lambda - \lambda z) - z)} \tag{10}$$

where $\tilde{\mathcal{B}}(s)$ is the Laplace Transform of the service time pdf, $\tilde{\mathcal{F}}(s)$ is the Laplace Transform of the first (exceptional) service time pdf, and m_F is the mean of the first service time.

Before we replace the M/G/1 generating function in equation (3) with the one given in equation (10) we will find $\tilde{\mathcal{B}}(s)$ in the context of the leaky bucket equation. As opposed to $\tilde{\mathcal{B}}(s)$ in Section 3.1.1, $\tilde{\mathcal{B}}(s)$ does not represent the service time seen by the packet buffer. However, it is purely the Laplace Transform of the packet interarrival time pdf since the initial residual time factor is taken care of by $\tilde{\mathcal{F}}(s)$ and m_F.

The token interarrival time is deterministic with mean $\delta = 1/\lambda_T$. The Laplace Transform of it's pdf is therefore

$\tilde{\mathcal{B}}(s) = e^{-\delta s}$

and

$\tilde{\mathcal{B}}(\lambda - \lambda z) = e^{\lambda\delta z - \lambda\delta} = e^{\rho z - \rho}$.

Using equation (10) in the equation for the combined queues (equation (3)) yields

$$Q(z) = 1 - (1 - Q(0) + Q(0)\Theta(0))\left(1 - \frac{(1-\rho)(e^{\rho z-\rho} - z\tilde{\mathcal{F}}(\lambda - \lambda z))}{(1+\lambda m_F - \rho)(e^{\rho z-\rho} - z)}\right). \tag{11}$$

Setting z to 0, the value of $Q(0)$ is found as

$$Q(0) = \frac{(1-\rho)}{2 - 2\rho + \lambda m_F - (1-\rho)\Theta(0)}$$

and equation (11) becomes

$$Q(z) = 1 - \left(1 - \frac{(1-\rho)}{2-2\rho+\lambda m_F - (1-\rho)\Theta(0)} + \frac{(1-\rho)\Theta(0)}{2-2\rho+\lambda m_F - (1-\rho)\Theta(0)}\right)$$
$$\left(1 - \frac{(1-\rho)(e^{\rho z-\rho} - z\tilde{\mathcal{F}}(\lambda - \lambda z))}{(1+\lambda m_F - \rho)(e^{\rho z-\rho} - z)}\right)$$

where $\mathcal{F}(s)$ and m_F are, respectively, the Laplace Transform and the mean of the pdf of the initial residual time. This initial residual time is the time a packet, arriving at an empty buffer, has to wait until a token arrives. It is described in detail in Section 4.1.

Chapter 4

Validation

To demonstrate the success of the analysis and the validity of equation (4), the mean queue length was derived from the equation. Values of the mean queue length were computed for different arrival rates and token pool sizes. A simulation model of the leaky bucket mechanism was designed. The mean queue length results from the analysis were then compared against the mean packet buffer queue length results from the simulation.

4.1 Mean Queue Length Analysis

In this section we will derive the mean queue length from the moment generating function of the number of packets in the buffer given in equation (4). Since we are finding the mean queue length we are interested in the time average behavior of the system.

The service time seen by a packet, as it waits for a token to arrive, is δ, the token interarrival time, if the packet queues into a non empty buffer. If it queues into an empty buffer, the packet will only see the remaining time since the last token arrived. This residual service time also depends on the last event that made both queues empty, namely, a token arrival taking the last queued packet or a packet arriving to take the last queued token.

The first packet arriving at an empty buffer, after being emptied by a token, arrives t seconds after the last token arrival with a probability

$$\Pr[\text{packet arrives } t \text{ seconds after the token arrival that made the queue empty}] = \lambda e^{-\lambda t}$$

since the arrival of packets is Poisson distributed and also due to the memory-less properties of Poisson arrivals. Of course we are only interested in packet arrivals before the next token arrives (otherwise the token pool will not remain empty), so the packet arrival probability density function (pdf) within the token interarrival time is normalized as follows

$$\Pr[\text{arrival at time } t \mid \text{arrival within } t = 0 \text{ to } \delta] = \frac{\lambda e^{-\lambda t}}{\int_0^{\delta} \lambda e^{-\lambda t} dt}.$$

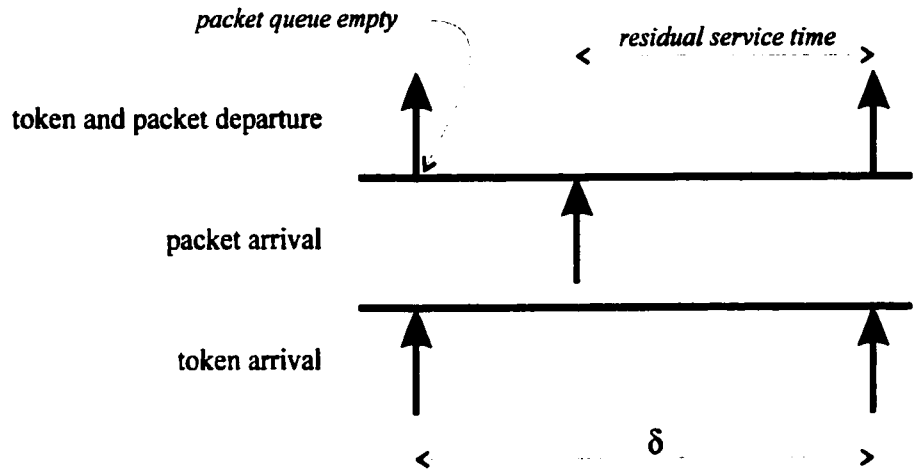


Figure 4 *Packet arrival to a buffer made empty by a token arrival*

On average, the first packet will be arriving at

$$\text{E[time of arrival]} = \frac{\int_0^{\delta} t\lambda e^{-\lambda t} dt}{\int_0^{\delta} \lambda e^{-\lambda t} dt} = \frac{\lambda e^{-\delta\lambda}\delta + e^{-\delta\lambda} - 1}{\lambda\left(e^{-\delta\lambda} - 1\right)} = \frac{\rho e^{-\rho} + e^{-\rho} - 1}{\lambda\left(e^{-\rho} - 1\right)}.$$

This packet then sees an average service time of

$$\text{E[service time when queue is empty]} = \delta - \frac{\rho e^{-\rho} + e^{-\rho} - 1}{\lambda\left(e^{-\rho} - 1\right)}.$$

Given that the last event leaving both queues empty was a packet arrival, a packet arriving to find an empty buffer will find yet a different service time. This service time can be calculated, as in the first case, by finding the probability of the time of arrival of the second packet within the token interarrival period. To simplify the computation and also since it is difficult to find the probability of a packet seeing one residual service time or the other, as it involves both queues, we will resort to approximate the residual service time as being of the first type only. This will be one of two approximations used in computing the mean queue length knowing that it will overestimate the service time as seen when comparing Figure 4 and Figure 5, hence slightly overestimating the mean queue length.

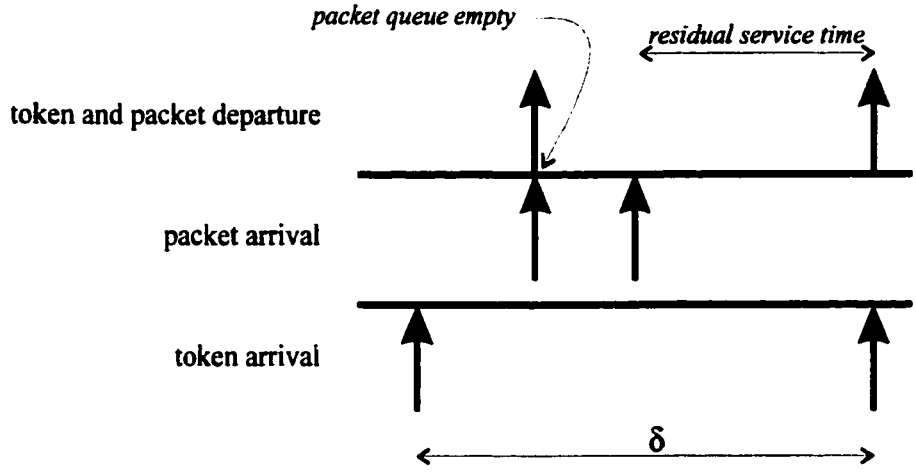


Figure 5 *Packet arrival to a buffer made empty by a packet arrival*

The service time seen by a packet whether it finds the buffer empty or not is now represented as a deterministic time with an interdeparture interval being equal to the average service time seen by a packet. In an M/G/1 queue the probability of a customer arriving to find a non-empty queue is equal to the utilization ρ (which is λ/λ_T in our case) and the probability of the customer finding an empty queue is $(1-\rho)$ [1]. The service time is the weighted average of the two possibilities. That is the average service time is equal to

ρx(service time seen by an arrival to a non empty buffer) + $(1-\rho)$(average service time seen by an arrival to an empty buffer)

or

$$\text{Average service time} = \rho\delta + (1-\rho)\left(\frac{\delta - \lambda e^{-\rho} + e^{-\rho} - 1}{\lambda\left(e^{-\rho} - 1\right)}\right)$$

$$= \frac{\rho\delta\lambda e^{-\rho} - \delta\lambda - e^{-\rho} + 1 + \rho e^{-\rho} - \rho}{\lambda\left(e^{-\rho} - 1\right)}$$

$$= \frac{\rho^2 e^{-\rho} - 2\rho - e^{-\rho} + 1 + \rho e^{-\rho}}{\lambda\left(e^{-\rho} - 1\right)}$$

Considering this average service time as a deterministic time its $\mathfrak{LT}$., $\mathfrak{B}(s)$ becomes

$$\mathfrak{B}(s) = e^{-\frac{\rho^2 e^{-\rho} - 2\rho - e^{-\rho} + 1 + \rho e^{-\rho}}{\lambda\left(e^{-\rho} - 1\right)} s}$$

and

$$\mathcal{B}(\lambda - \lambda z) = e^{-\frac{\rho^2 e^{-\rho} - 2\rho - e^{-\rho} + 1 + \rho e^{-\rho}}{\lambda\left(e^{-\rho} - 1\right)}(\lambda - \lambda z)} = e^{\frac{\rho^2 e^{-\rho} - 2\rho - e^{-\rho} + 1 + \rho e^{-\rho}}{e^{-\rho} - 1}(z-1)}.$$

Plugging this result into equation (4) it yields

$$Q(z) = 1 - \left(1 - \frac{1-\rho}{1-\rho+\rho\Theta(0)} + \frac{(1-\rho)\Theta(0)}{1-\rho+\rho\Theta(0)}\right)\left(1 - \frac{e^{\frac{\rho^2 e^{-\rho} - 2\rho - e^{-\rho} + 1 + \rho e^{-\rho}}{e^{-\rho} - 1}(z-1)}(1-\rho)(1-z)}{e^{\frac{\rho^2 e^{-\rho} - 2\rho - e^{-\rho} + 1 + \rho e^{-\rho}}{e^{-\rho} - 1}(z-1)} - z}\right).$$

The mean queue length can be found by evaluating the derivative of $Q(z)$ at $z = 1$. The derivative gives an indeterminate result at $z = 1$ and requires the use l'Hospital's rule multiple times. Instead, the Maple V [17][18] function `dQ:=diff(Q(z), z)`, is used to find dQ, the derivative $\frac{\partial Q(z)}{\partial z}$, and the function `limit(dQ, z=1)` to find the limit of the derivative as z goes to 1. The result of the limit is the mean queue length

$$\text{Mean queue length} = \left.\frac{\partial Q(z)}{\partial z}\right|_{z=1}$$
$$= \frac{1}{2}\frac{\Theta(0)\left(\rho^4 - 4\rho^3 e^{\rho} + 2\rho^3 - 8\rho e^{2\rho} + 12\rho e^{\rho} + 4\rho^2 e^{2\rho} - 4\rho - 6e^{\rho} + 3 - 3\rho^2 + 3^{2\rho}\right)}{\left(\rho + 2e^{\rho} - 2\rho e^{\rho} + \rho^2 - 2\right)\left(\rho - 2e^{\rho} + 2\right)\left(1 - \rho + \rho\Theta(0)\right)} \quad (12)$$

4.2 D/M/1/*K* Computations

Due to the difficulty in finding a closed form solution by solving the D/M/1/*K* equations for an unknown *K* the equation is validated by solving the D/M/1/*K* using Maple V [17][18].

The Maple V function `x:=linsolve(A,b)` solves a set of linear equations described by $\boldsymbol{A}$ (a matrix of size nxm), $\boldsymbol{b}$ (a column vector of size n) and $\boldsymbol{x}$ (a column vector of size m) that meet the relation $\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}$ where $\boldsymbol{x}$ is the unknown.

To solve $\boldsymbol{\pi}\,\boldsymbol{P} = \boldsymbol{\pi}$ with `linsolve` some transformation is required. First the unknown row vector $\boldsymbol{\pi}$ is moved to the left-hand side of the equation. The result is $\boldsymbol{\pi}\,(\boldsymbol{P}-\boldsymbol{I}) = \boldsymbol{0}$ where $\boldsymbol{I}$ is the Identity matrix and $\boldsymbol{0}$ is an all-zero vector. Then, since `linsolve` solves

the set of equations of the form $\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}$ we must take the transpose of the $(\boldsymbol{P}\text{-}\boldsymbol{I})$ matrix and the column form of $\boldsymbol{\pi}$ to satisfy $(\boldsymbol{P}\text{-}\boldsymbol{I})^{\mathrm{T}}\,\boldsymbol{\pi}^{\mathrm{T}} = \boldsymbol{0}$. The last step is to add the normalization equation to the matrix. The result is then an equation of the form $\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}$ which is defined as

$$\begin{bmatrix} 1-b_0-1 & 1-\sum_{k=0}^{1} b_k & 1-\sum_{k=0}^{2} b_k & \cdots & 1-\sum_{k=0}^{K-1} b_k & 1-\sum_{k=0}^{K-1} b_k \\ b_0 & b_1-1 & b_2 & \cdots & b_{K-1} & b_{K-1} \\ 0 & b_0 & b_1-1 & \cdots & b_{K-2} & b_{K-2} \\ 0 & 0 & b_0 & \cdots & b_{K-3} & b_{K-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & b_0 & b_0-1 \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{bmatrix} \begin{bmatrix} \pi_0 \\ \pi_1 \\ \pi_2 \\ \vdots \\ \pi_{K-1} \\ \pi_K \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}.$$

To obtain π_0 for different sizes B of the token pool the above equation must be solved for $K = B$. As an example, for token pool size $B = 2$ the equation is

$$\begin{bmatrix} 1-b_0-1 & 1-b_0-b_1 & 1-b_0-b_1 \\ b_0 & b_1-1 & b_1 \\ 0 & b_0 & b_0-1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \pi_0 \\ \pi_1 \\ \pi_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

A Maple V function called `leaky_bucket` was written to calculate the mean queue length given B, the token pool size, and ρ, the normalized arrival rate (refer to Appendix I). Although this function can be used to generate a continuous plot of the mean queue length with respect to the normalized arrival rate ρ using the Maple V function `plot`, only a few points are of interest to us. These points are used to compare the results of the analytical approximation against the simulation results.

4.3 Simulation

The simulation part of the validation is performed with the help of the OPtimized Network Engineering Tools (OPNET). OPNET is a workstation-based environment for the modeling and simulation of communication systems, protocols, and networks.

OPNET features include: graphical specifications of models; a dynamic event-scheduled Simulation Kernel; integrated data analysis tools; and hierarchical object-based modeling [19].

In this section the model is described briefly and the code is provided in Appendix II. The following description uses some OPNET terms like: node; process; generator; queue; processor; interrupt... etc. Details on the operation of the tool and the true meaning of these terms can be found in the OPNET documentation [19].

The simulation model is very simple. It consists of the OPNET nodes depicted in Figure 6: a token source, `t_source`; a token queue, `t_queue`; a customer (or packet) source, `c_source`; a customer queue, `c_queue`; and three processors (called: `server`, `network` and `statistics`).

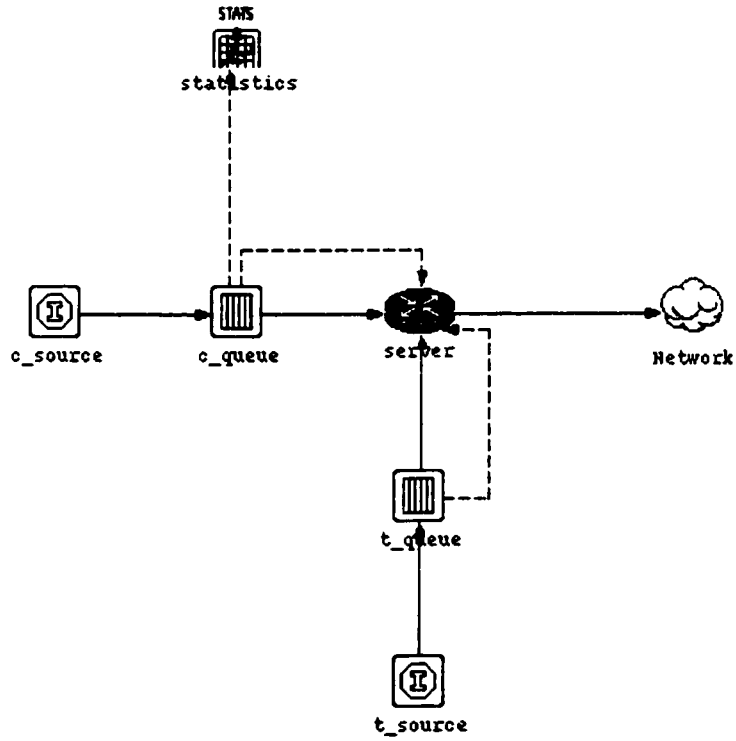


Figure 6 *Node diagram of the OPNET leaky bucket model*

The token source is an OPNET ideal generator. It generates tokens (which are normal OPNET packets distinguished from the customer packets by the stream they follow) with

a deterministic interarrival time of one second (i.e. λ_T = 1sec). This one second is used to make the normalized packet arrival rate ρ equal to the packet arrival rate in the simulation. Any other value can be used as long as ρ is calculated as λ/λ_T.

The token queue is also the OPNET built in queue. It is configured as a FIFO queue with a limited size, B, which can be assigned at simulation time.

The packet source is also an ideal generator but it is configured for a Poisson interarrival time with mean rate λ seconds which is equal to ρ since λ_T = 1 second.

The packet queue is configured as a FIFO with infinite size.

The network processor is basically a sink, which destroys packets.

The statistics processor is custom designed to calculate the time average mean packet queue length. This processor is required since the built in mean queue length function of OPNET only calculates the mean queue length as seen by an arrival or a departure to and from the queue respectively.

The server processor is the heart of the model. It is a custom designed process that removes a packet and a token from their respective queues if both exist, destroys the token and forwards the packet to the network. The server does this function by waiting for a statistics interrupt from either one of the two queues. This interrupt is generated every time the number of packets in the queue increases by one. Once interrupted the server looks for the number of packets in each queue. If both queues are not empty it sets a self-interrupting timer for the duration of the service time. The service time, although set to zero and not used in our simulation, was added to allow the model to simulate a non-zero service time for future use. Once the service time timer expires the server removes a packet from the token queue and destroys it and removes a packet from the packet queue and forwards it to the network (which in this model also destroys the packet). The server

then looks again for any packets or tokens that arrived during the service time and services them if needed.

The details of the simulation model and its code are provided in Appendix II.

4.4 Results

Simulation runs were executed for different values of B, the token pool size, and for ρ, the normalized packet arrival rate. In addition, for each point, four simulation runs were executed using different random seeds in order to find a confidence interval, in which the real result curve will fall. These runs are considered observations of a random experiment.

The Student's t function [16] is used to find the confidence interval for each point from the results of the four simulation runs. The results are plotted using the Student's t equation $\bar{x} \pm t_{\alpha/2}(n-1)s/\sqrt{n}$ for a 95% confidence interval where:

- $\bar{x}$ is the estimated mean, found by calculating the mean of the four observations;
- s^2 is the estimated variance, also found by calculating the variance of the four observations;
- $1\text{-}\alpha$ is the confidence coefficient for a $100(1\text{-}\alpha)\%$ confidence interval (i.e. for a 95% confidence interval $\alpha = 0.05$);
- n is the number of observations which is 4 in our case;
- and $t_\alpha(r)$ is the $100(1\text{-}\alpha)$ percentile of the t distribution with r degrees of freedom which can be found in statistic tables such as the one in [16].

The resulting curve with its 95% confidence interval is then plotted in Figure 7 against the analytical approximation results of equation (12).

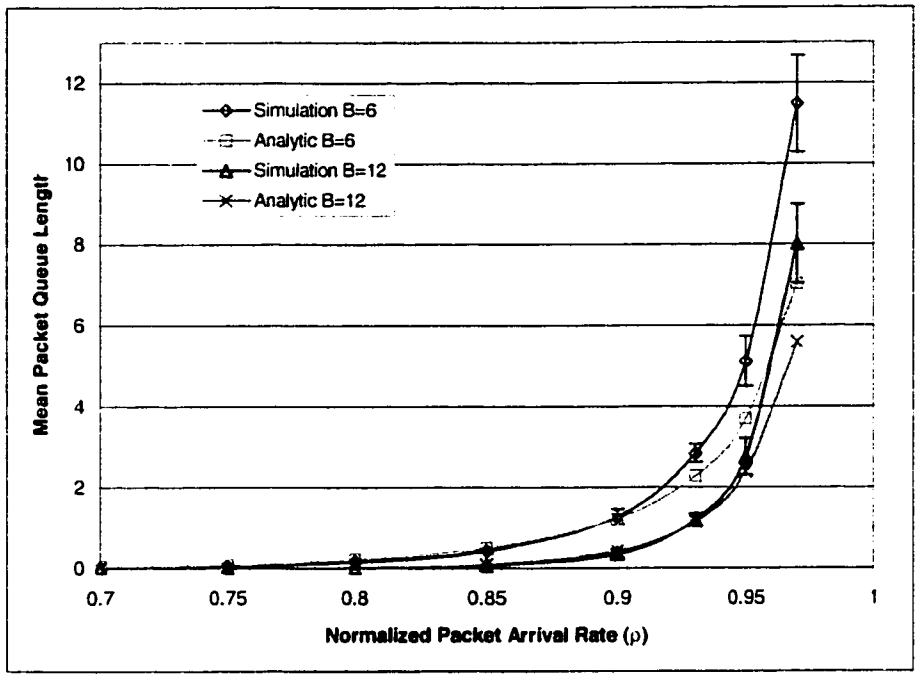


Figure 7 *Simulation and Analysis Results for Token Pool Size, B, of 6 and 12.*

4.5 Discussion of the results

Figure 7 shows the mean queue length as found from the simulation and the analytical approximation for a token pool size of $B = 6$ and 12 and for a normalized packet arrival rate $\rho = 0.7$ to 0.97. The values of ρ were selected to be below 1 for stability of the system. Values below 0.7 were not plotted on the graph as they are very close to zero and will not add any value at the scale of this plot.

Let us start by commenting on the shape of the curve in general. As we can see from the graph, the mean queue length starts very low for low values of ρ, which is the ratio of the packet arrival rate λ to the token generation rate λ_T. As ρ, the normalized packet arrival rate, increases, more and more tokens are removed from the token pool and the probability of depleting the tokens becomes higher, therefore, the packet buffer starts to buildup. We can also see that as ρ goes higher the buffer starts to buildup faster.

By comparing the two plots, for $B = 6$ and 12 we can see the effect of the token pool size B as it is doubled. As the token pool size increases the expected packet buffer queue

length decreases. This is due the capability of the token pool to hold more tokens, which will allow more packets to enter the network and therefore prevent them from queueing. If we compare the ratio of change caused by the change in the token pool size, B, we see that the expected queue length changes dramatically for low values of ρ but the difference becomes less and less important as ρ increases. This also can be explained as when the normalized packet arrival rate is high, the token pool has less chance of filling up completely and therefore the size of the pool plays a less important role in holding tokens. Therefore, B's effect on the packet buffer length is reduced.

The next step is to compare the analytical results to the simulation results. From Figure 7, the two curves seem to follow each other generally. For low values of ρ the difference is not apparent but it is more pronounced for ρ grater than 0.9 and 0.95 for B = 6 and 12 respectively. Although it does not fall within the confidence interval of the simulation, the deviation can be explained by the two approximations taken in the computation of the mean queue length, namely the D/M/1/K analysis and the residual service time simplification.

As explained in Section 4.1 the residual service time simplification assumed that when a packet is queued into an empty buffer the buffer had always been emptied by a token arrival. This is considered an overestimation of the residual service time and, hence, accounts for the slight overestimation of the mean queue length. This is apparent in the lower values of ρ. For high values of ρ, however, the two curves cross and the approximation starts to underestimate the mean queue length. This is clearly due to the D/M/1/K analysis.

The D/M/1/K is analyzed at the embedded Markov points. These discrete-time points are set at the token arrival instants and, therefore, describe the state of the token pool as seen by arriving tokens only. As ρ increases, the possibility of the token pool becoming empty increases. Since the embedded Markov points are at the end of the token interarrival period the probability of finding the pool empty is amplified by the D/M/1/K results.

Figure 8 shows an example of such a situation where ρ is high and the packet arrival rate is very close to the token arrival rate. As this example shows, the embedded Markov points see an empty queue for the duration of four interarrival periods when in reality the time average probability of the queue being empty is much lower.

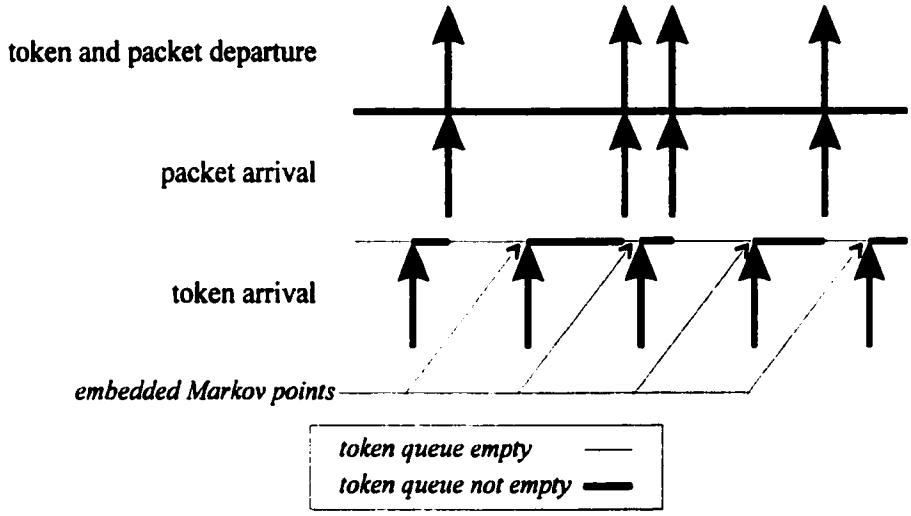


Figure 8 *Example of the effect of the embedded Markov points on the results of the D/M/1/K analysis*

As ρ increases the embedded Markov point analysis of the D/M/1/K queue amplifies more and more the probability of the token pool being empty. This is apparent in the graph of Figure 7 as the deviation of the analytical curve from the simulation curve increases with ρ. In addition, as the size of the token pool B increases we can see that the point where the two curves cross is moved to the right and the difference between the curves become less pronounced for the same values of ρ. This is because as the pool size increases, the probability of the pool being empty is reduced, for the same values of ρ, and the possibility of overestimating this probability is also reduced. As B grows even higher, one would expect the approximation to be closer to the real curve and the deviation at high values of ρ to become less noticeable.

Chapter 5

Performance Evaluation

The results found in the previous section are put to work in the following sections. We will, first, evaluate the effect of the arrival rate and of the token pool size on the mean queue length and the mean waiting time in the packet buffer. We will also study the effect of the system parameters on the loss rate of the finite storage leaky bucket.

5.1 Mean Queue Length

The infinite buffer leaky bucket model is characterized by the packet arrival rate λ, the token arrival rate λ_T, and the token pool size B. Using the Maple V function `leaky_bucket` listed in Appendix I, the mean queue length is plotted with respect to the normalized packet arrival rate ρ, where $\rho = \lambda/\lambda_T$. By changing the value of B, a family of curves is obtained. These curves are plotted in both the linear and the logarithmic scale in Figure 9 and Figure 10 respectively.

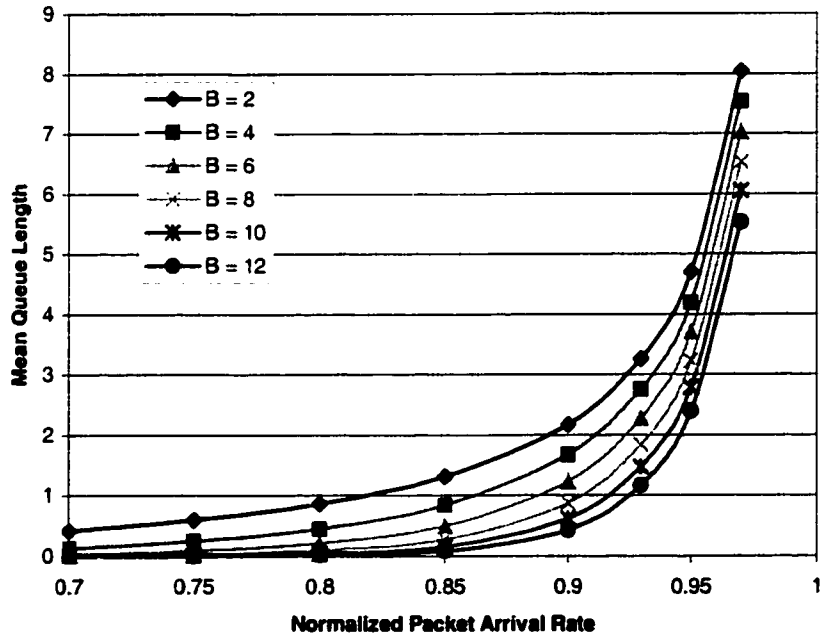


Figure 9 *Mean queue length vs. arrival rate for different values of B. (Linear scale)*

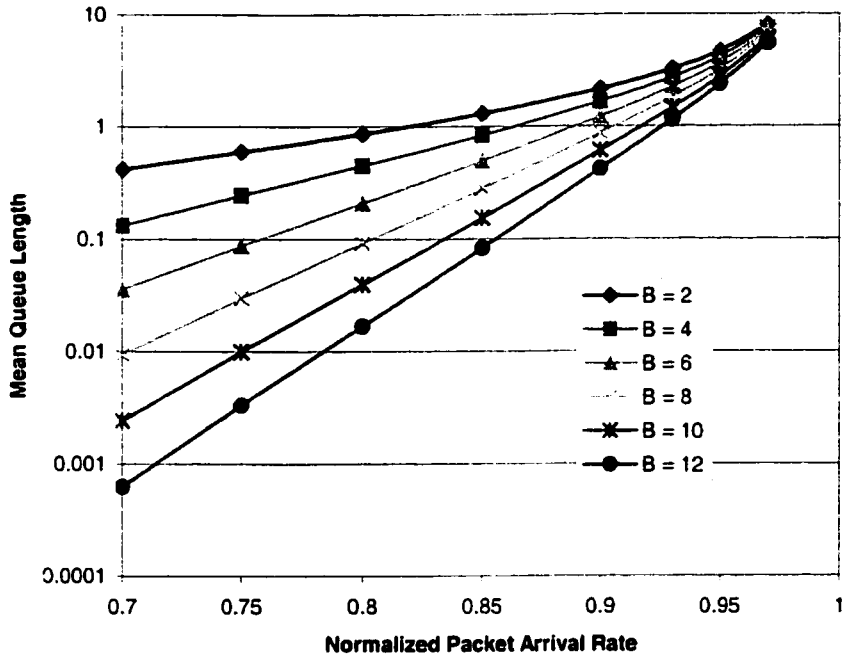


Figure 10 *Mean queue length vs. arrival rate for different values of B. (Logarithmic scale)*

From the plots, above, one can study the effect of the system parameters on the mean queue length. It can be clearly seen that the mean queue length increases exponentially with the arrival rate. As the token pool size increases, the mean queue length decreases. This is because the higher number of tokens that can be stored allows more packets to enter the network without having to queue. The effect of the token pool size becomes less evident as the arrival rate increases, since the pool will less likely be allowed to overflow and the size of the pool plays a less important role in the system. This can be shown clearly when the mean queue length is plotted against the token pool size as we do in Figure 11 that follows.

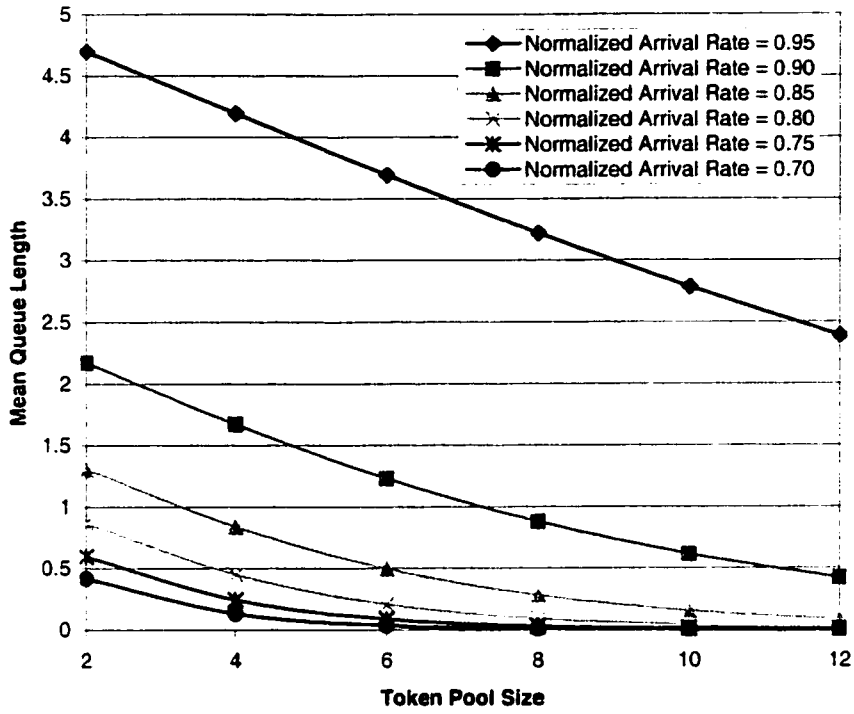


Figure 11 *Mean queue length vs. token pool size for different values of ρ.*

5.2 Mean Waiting Time

The mean waiting time is calculated, using the Little's formula, from the values of the mean queue length in the previous section. Little's equation calculates the mean waiting time, as seen by an arriving packet, as $W = \frac{L}{\lambda}$, where W is the mean waiting time, L is the mean queue length and λ is the arrival rate.

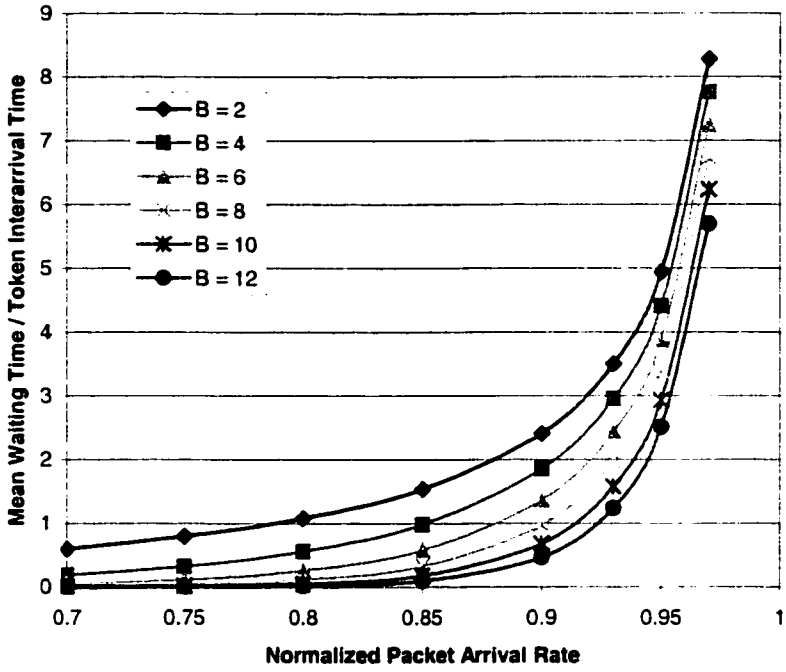


Figure 12 *Normalized mean waiting time as seen by arriving packets.*

As expected, the mean waiting time family of curves follows the general trend of the mean queue length. Since the mean queue length depends on ρ, which we call the normalized packet arrival rate, and the mean waiting time depends on λ in addition to ρ, the plots of Figure 12 represent the mean waiting time normalized to the token interarrival time. In order to find the real mean waiting time, the value in the plot is simply multiplied by the token interarrival time. The result will have the same time unit as the token interarrival time.

5.3 Packet Loss Probability

The limitation to the number of packets that can be queued in the finite storage leaky bucket model creates the possibility of losing packets when the buffer is full. The measure of the loss rate is a very important factor in studying the performance of the system since it has to be kept to acceptable levels for the type of traffic going through the network.

The Maple V function `LossProb` is used to compute the probability that a packet arrives to find a full buffer and, thus, be lost.

In Figure 13 the loss probability is plotted with respect to the normalized packet arrival rate, ρ, for a fixed token pool size, B, and variable packet buffer size, K. The result is a family of curves showing the effect of the packet buffer on the loss probability.

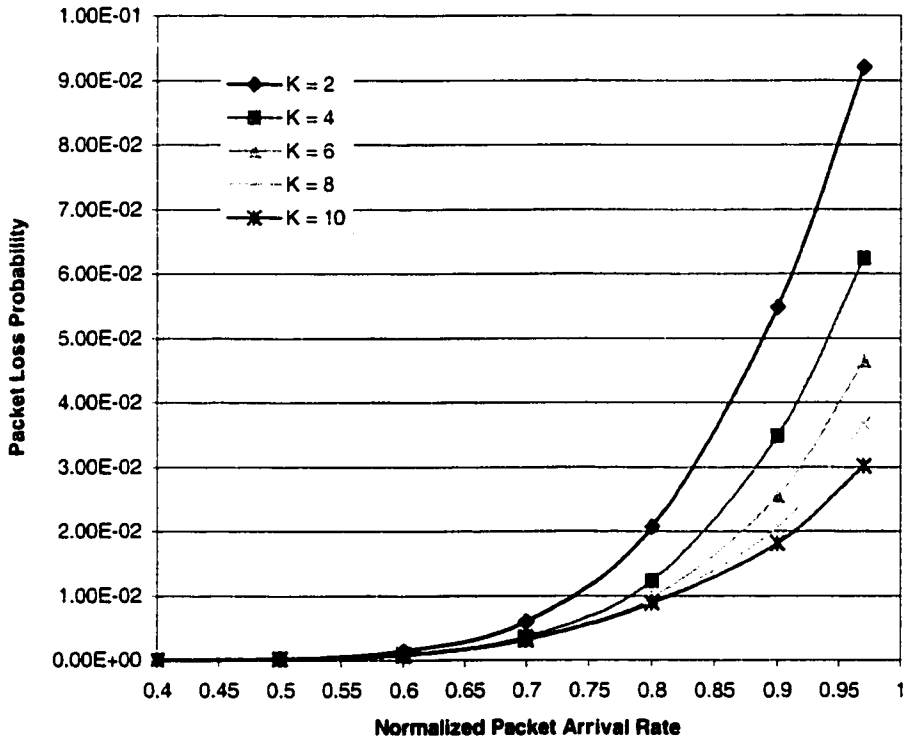


Figure 13 *Packet loss probability for B = 6 and different values of K.*

In Figure 14 the loss probability is plotted with respect to the normalized packet arrival rate, ρ, for a fixed packet buffer size, K, and variable token pool size, B. The result is a family of curves showing the effect of the token pool size on the loss probability.

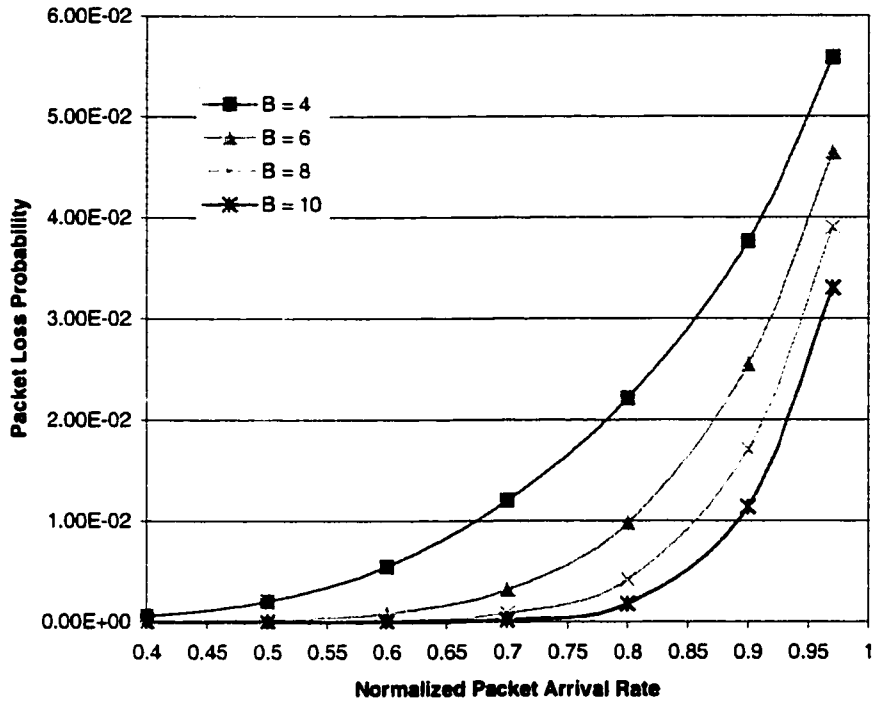


Figure 14 *Packet loss probability for K = 6 and different values of B.*

From the plots of Figure 14 it is clear that the effect of the token pool is less noticeable at higher packet arrival rates than at the low rates. Echoing what has been said for the mean waiting time in the finite buffer system, this is explained that at high packet arrival rates the pool will less likely be allowed to overflow. If the pool is not full, or almost empty, all the time its size will definitely play a less important role in the system.

In contrast, the results of Figure 13 show that the effect of the packet buffer size, *K*, is more important at high packet arrival rate and less noticeable at low arrival rate. This can be explained that the probability of the packet buffer reaching it's capacity is increased by the packet arrivals. The size of the buffer is therefore critical in affecting the performance in this situation as opposed to when the packet arrival rate is low.

The loss probability data, shown above, is presented once more in Figure 15 and Figure 16 with the packet buffer size and the token pool size on the horizontal axis respectively.

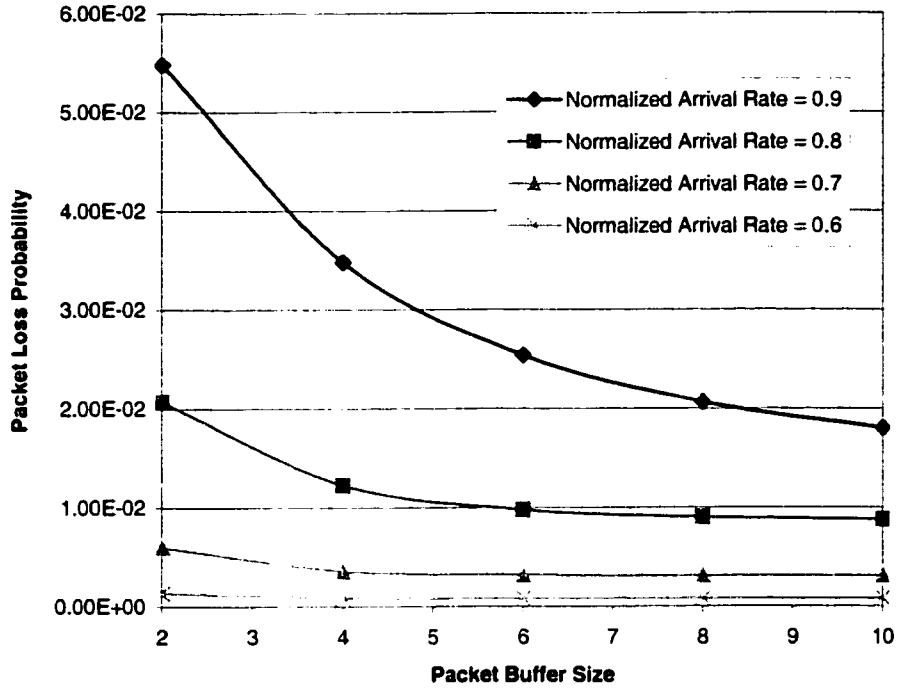


Figure 15 *Packet loss probability vs. packet buffer size, K, for B = 6.*

The graph, above, shows the effect of the packet buffer size on the packet loss probability for fixed values of the normalized packet arrival rate, ρ. The loss probability is reduced exponentially when the packet buffer size increases. It is evident from this plot that for larger values of ρ the effect of the packet buffer size is more noticeable, as discussed earlier.

Similarly, Figure 16 shows the effect of the token pool size on the packet loss probability. By examining the individual curves for the different values of ρ, the following statement becomes evident. Although both B and K, as they increase, are reducing the loss probability exponentially, each of the two is doing the opposite of the other with respect to ρ. The curves for the various values of ρ, in Figure 15, are initially far apart but start to converge as K increases. This is because K affects the rate of change in the loss probability curve for high values of ρ more than for the low values. In Figure 16, on the other hand, the different curves start very close but their separation increases as B is increased.

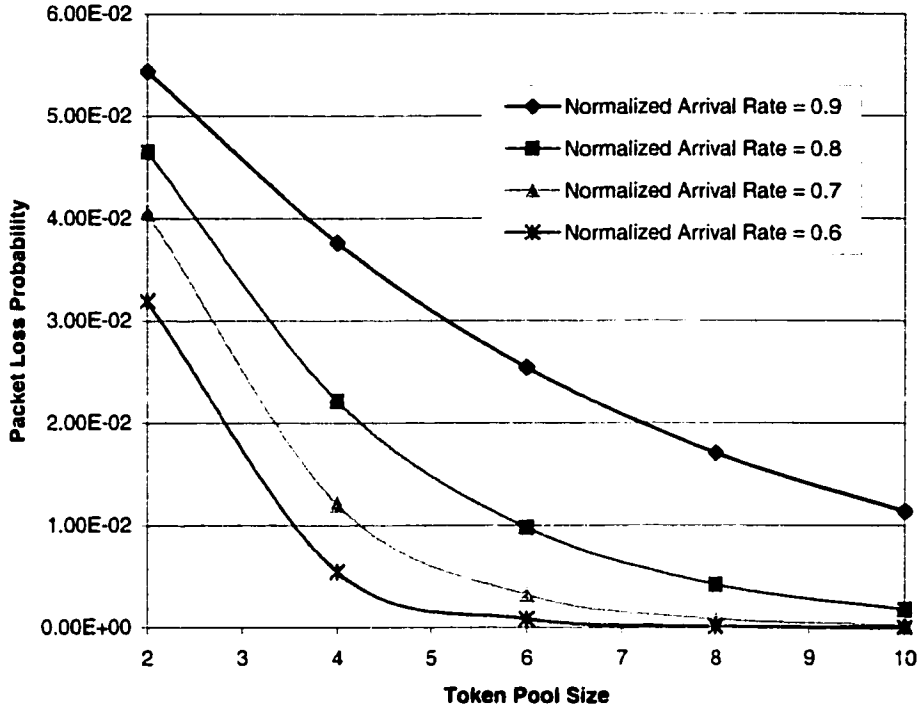


Figure 16 *Packet loss probability vs. token pool size, B, for K = 6.*

Chapter 6

Engineering Applications

The performance analysis of the leaky bucket and the results found in this thesis may eventually be used in engineering real networks. The simulation or the Maple V functions will aid the network engineer in selecting the proper leaky bucket parameters to avoid congestion while maintaining the required quality of service. The parameters that were plotted in this thesis are arbitrary but do not limit the application of the results. In fact, any value for ρ can be used as long as it is less than 1 for the system to be stable. The values of B and K used here range between 2 and 12 but higher values can be used if required. However, high values of the parameter B will be limited by the memory size and execution time when used in the Maple V functions since it affects the size of the set of linear equations to be solved.

6.1 Design Guidelines

This section presents some guidelines for the use of the performance evaluation results of this thesis. These guidelines will aid the reader in selecting the parameters to plug into the Maple V functions of Appendix I or to find the relevant performance points from the graphs presented earlier.

With the exception of the mean waiting time, the packet arrival rate and the token arrival rate do not individually affect the outcome of the analysis. It is the ratio of these two parameters that has to be used instead. This fact reduces the complexity of using the graphs of the mean waiting time and of the loss rate since one curve summarizes the results regardless of the values of λ and λ_T individually.

To find the mean queue length in the infinite storage leaky bucket, one must first find ρ, the ratio of the packet arrival rate to the token arrival rate. The second parameter is the

token pool size. Both of these values are simply inserted into the `leaky_bucket` function of Appendix I. The mean queue length is directly calculated by the function.

Figure 9 and Figure 10 show the plots of the mean queue length with respect to ρ for different values of the token pool size, B. If the required token pool size is one of the values used in these graphs the mean queue length is directly obtained from the curves. If it is between two other values, the results can be approximated by interpolation.

The mean waiting time requires the use of the packet arrival rate parameter in addition to ρ and B. After finding the mean queue length, as described above, Little's rule is used. The mean queue length is divided by the packet arrival rate. The mean waiting time is the result of this division. The time unit is obtained from the packet arrival rate.

In Figure 12 some pre-calculated values of the mean waiting time have been plotted. Since these curves are plotted with respect to ρ, for simplicity, the y-axis represent the mean waiting time normalized to the token interarrival time. After finding the y-axis value, for the given ρ, it should be multiplied by the token interarrival time. In other words, the value obtained from Figure 12 is divided by λ_T to calculate the mean waiting time.

In the finite buffer system a new parameter is added to the picture. The packet buffer size, K, must be used, in addition to B and ρ. These three parameters are simply plugged into the `LossProb` function of Appendix I to find the loss probability of this system. The curves of selected values of these parameters were plotted in Figure 13 and Figure 14. Since these plots depend on three parameters, one of the parameters had to be fixed in each plot while the other varied. This will definitely limit the application of these plots. The `LossProb` function would be a better tool for it's capability of covering all required values.

6.2 Application Example for the Mean Queue Length and Waiting Time

To illustrate the application of the mean queue length and the mean waiting time results we will consider the following example:

Suppose we have a leaky bucket mechanism used to control a stream of packets with an arrival rate of 200 packets per second. The leaky bucket is fed by tokens at the rate of 300 tokens per second. The tokens are stored in a pool capable of holding 4 tokens. Using the analysis carried out in this thesis, the following parameters are inserted into the `leaky_bucket` function of Appendix I: $\rho = \lambda/\lambda_T = 200/300$ and $B = 4$. The function returns 0.8736964885. Since this is an approximation and knowing that, with a value of $\rho = 0.666$, this approximation slightly over estimates the results we can say that the average packet buffer length to be expected in this system is slightly lower than 0.874 packets.

The mean waiting time seen by an arriving packet can also be calculated from the mean queue length using Little's Formula: $W = \frac{L}{\lambda}$, where W is the mean waiting time, L is the mean queue length and λ is the arrival rate. The mean waiting time seen by an arriving packet in our example is then approximated at W = 0.874/200 = 0.00437 (second) = 4.37 (ms).

Chapter 7

Conclusion

In conclusion, by analyzing the mutual dependence between the packet and the token queues we were able to come up with the generating function of the packet buffer length. This generating function describes the packet buffer queue length distribution completely and can be used to tell us a lot about the system behavior.

The validity of the equation was demonstrated. This was achieved by finding an approximation of the mean queue length, from the generating function, and computing it for different values of the normalized packet arrival rate and the token pool size. The results were compared against those obtained from the simulation. The mean queue length, as well as the mean waiting time, were plotted against the system parameters in order to describe the performance of the system.

The capability of this method to analyze variations of the system was demonstrated by using M/G/1/K instead of M/G/1 results for the packet buffer in order to study a lossy system. This was also done by analyzing the system with a bulk arrival assumption.

7.1 Future work

The work done in this thesis represents the first step towards finding a complete model capable of analyzing variants of the system – a non-zero service time for instance. The non-zero service time is more challenging to analyze. This is because it allows the queues to be non empty at the same time. However, we believe it can be analyzed, in the future, using the same approach.

In addition, the approximation of the mean queue length can be improved by analyzing the D/M/1/K queue differently so that the results describe the queue not only as seen by arriving tokens. This change, alone, will play a major role in improving the accuracy of

the mean queue length found in this thesis, since it accounts for the worse error at high values of the normalized packet arrival rate.

References

[1] L. Kleinrock, "**Queueing Systems**", Volume 1, John Wiley & Sons, 1975.

[2] T. Takine, B. Sengupta and T. Hasegawa, "**A Conformance Measure for Traffic Shaping in High-Speed Networks with an Application to the Leaky Bucket**", *Proceedings IEEE INFOCOM '94*, Volume 2, p. 474-481, IEEE 1994.

[3] D. Logothetis and K. Trivedi, "**Transient Analysis of the Leaky Bucket Rate Control Scheme Under Poisson ON-OFF Sources**", *Proceedings IEEE INFOCOM '94*, Volume 2, p. 490-497, IEEE 1994.

[4] J. Turner, "**New Directions in Communications (or Which Way to the Information Age)**", *IEEE Communications Magazine*, Volume 24, no. 10, p. 8-15, October 1986.

[5] D. Lee, "**Effects of Leaky Bucket Parameters on the Average Queuing Delay: Worst Case Analysis**", *Proceedings IEEE INFOCOM '94*, Volume 2, p. 482-489, IEEE 1994.

[6] H. Ahmadi, R. Guérin and K. Sohraby, "**Analysis of Leaky Bucket Access Control Mechanism with Batch Arrival Process**", Proceedings *of GLOBECOM'90*, p. 344-349, IEEE 1990.

[7] M. Sidi, W. Liu, I. Cidon and I. Gopal, "**Congestion Control Through Input Rate Regulation**", *Proceedings of GLOBECOM'89*, p. 1764-1768, IEEE 1989.

[8] G. Wu and J. Mark, "**Discrete Time Analysis of Leaky-Bucket Congestion Control**", *Computer Networks and ISDN Systems 26 (1993)*, p. 79-94, North-Holland 1993.

[9] V. Anantharam and T. Konstantopoulos, "**Burst Reduction Properties of the Leaky Bucket Flow Control Scheme in ATM Networks**", *IEEE Transactions on Communication*, Volume 42, Issue 12, p. 3085-3089, December 1994.

[10] D. Holtsinger and H. Perros, "**Performance of the Buffered Leaky Bucket Policing Mechanism**", *High-Speed Communication Networks. Proceedings of TriComm '92*, p. 47-69, Plenum 1992.

[11] S. Vamvakos and V. Ananthram, "**On the Departure Process of a Leaky Bucket System with Long-Range Dependent Input Traffic**", *IEEE ATM '97 Workshop Proceedings*, p. 223-232, IEEE 1997.

[12] S. Hairong and L. Lemin, "**Performance Analysis of Leaky Bucket Algorithm with Bursty Traffic Input in ATM Networks**", *Proceedings of ICCC '95. (12th International Conference on Computer Communication)*, p. 136-141, IOS Press 1995.

[13] S. Wittevrongel and H. Bruneel, "**Output Traffic Analysis of a Leaky Bucket Traffic Shaper Fed by a Bursty Source**", *SUPERCOMM/ICC '94*, Volume 3, p. 1581-1585, IEEE 1994.

[14] L. Kuang, "**Monotonicity Properties of the Leaky Bucket**", *Proceedings of the 31st IEEE Conference on Decision and Control*, Volume 1, p. 1014-1015, IEEE 1992.

[15] H. Takagi, "**Queuing Analysis, A Foundation of Performance Evaluation**", Volume 1, North-Holland 1991.

[16] R. Hogg and E. Tanis, "**Probability and Statistical Inference**", Macmillan Publishing Company 1988.

[17] K. Heal, M. Hansen and K. Rickard, "**Maple V, Learning Guide**", Spriger-Verlag 1996.

[18] M. Mongan, K. Geddes, K. Heal, G. Labahn, and S. Vorkvetter, "**Maple V, Programming Guide**", Springer-Verlag 1996.

[19] **OPNET 4.0 User Documentation**, Volumes 1-12, MIL 3 Inc. 1996-1997.

[20] H. Kobayashi, "**Modeling and Analysis. An Introduction to System Performance Evaluation Methodology**", Addison-Wesley, 1978.

[21] P. Harrison and N. Patel, "**Performance Modelling of Communication Networks and Computer Architectures**", Addison-Wesley, 1993.

[22] D. Gross and C. Harris, "**Fundamentals of Queueing Theory**", Second edition, John Wiley & Sons, 1985.

Appendix I: Maple V Code

This appendix contains the listing of the Maple V functions.

I.1 Code for Computing the Mean Queue Length

The first function is called `leaky_bucket`. It is used to calculate the mean queue length of the infinite buffer leaky bucket given the token pool size and the normalized arrival rate.

```
leaky_bucket := proc(B, r)    #Return the mean packet queue
                              #length given B, the token pool
                              #size, and r, the normalized
                              #packet arrival rate.
   local _row, column, i, k, b, pi_zero, P, v, x, mean;
                              #Variable declarations.

   b := proc(k, r)            #Find Pr[k services during an inter-
                              #arrival time].

     if k < 0 then
       0;
     else
       exp(-r) * r^k / k!;    #Poisson packet arrival.
     fi;
   end;

   P := matrix(B+2, B+1);     #Initialize the matrix size.
   v := vector(B+2);          #Initialize the vector size.
   x := vector(B+1);
   #Fill up the matrix P with the transpose of the transition
   #matrix and the normalization factors in row B+2.
   for _row from 1 to B+2 do
     for column from 1 to B do
       if _row = 1 then
         P[_row, column] := 1 - add(b(k, r), k = 0 .. column-1);
       elif _row = B+2 then
         P[_row, column] := 1;
       else
         if column - _row +1 < 0 then
           P[_row, column] := 0;
         else
           P[_row, column] := b(column - _row +1, r);
         fi
       fi;
     od;
     P[_row, B+1] := P[_row, B];
   od;

   #Subtract the Identity matrix from the transition matrix.
   for i from 1 to B+1 do
     P[i,i] := P[i,i] - 1;
   od;
```

```
    #Initialize the vector.
    for i from 1 to B+1 do
      v[i] := 0;
    od;
    v[B+2] := 1;

    with(linalg);
    #Solve the set of linear equations defined by Px=v,
    #where x=[pi_zero, pi_one, ... pi_B]
    x := linsolve(P, v);
    pi_zero := x[1];

    #Find the mean queue length by plugging the result
    #of pi_zero into the following equation.

    mean := 1/2*pi_zero*(-4*r^3*exp(r)+4*r^2*exp(2*r)+
          r^4+2*r^3-3*r^2+12*exp(r)*r-4*r-8*r*exp(2*r)-
          6*exp(r)+3+3*exp(2*r))/(2*exp(r)+r^2-
          2*exp(r)*r+r-2)/(r-2*exp(r)+2)/(1-r+r*pi_zero);

    #Return the value of the mean queue length.
    mean;

end:
```

I.2 Code for Computing the Loss Probability

The following function DM1K_empty is derived from the previous one and it calculates the probability that the D/M/1/*K* queue is empty.

```
DM1K_empty := proc(B, r)     #Return the probability that an
                              #M/D/1/B queue is empty given B
                              #and r -- the normalized departure
                              #rate.
 local _row, column, i, k, b, pi_zero, P, v, x, mean;
                              #Variable declarations.

   b := proc(k, r)            #Find Pr[k services during an inter-
                              #arrival time].

     if k < 0 then
       0;
     else
       exp(-r) * r^k / k!;    #Poisson packet arrival.
     fi;
   end;

   P := matrix(B+2, B+1);     #Initialize the matrix size.
   v := vector(B+2);          #Initialize the vector size.
   x := vector(B+1);
   #Fill up the matrix P with the transpose of the transition
   #matrix and the normalization factors in row B+2.
   for _row from 1 to B+2 do
     for column from 1 to B do
       if _row = 1 then
         P[_row, column] := 1 - add(b(k, r), k = 0 .. column-1);
       elif _row = B+2 then
         P[_row, column] := 1;
       else
         if column - _row +1 < 0 then
           P[_row, column] := 0;
         else
           P[_row, column] := b(column - _row +1, r);
         fi
       fi;
     od;
     P[_row, B+1] := P[_row, B];
   od;

   #Subtract the Identity matrix from the transition matrix.
   for i from 1 to B+1 do
     P[i,i] := P[i,i] - 1;
   od;

   #Initialize the vector.
   for i from 1 to B+1 do
     v[i] := 0;
   od;
   v[B+2] := 1;

   with(linalg);
```

```
  #Solve the set of linear equations defined by Px=v,
  #where x=[pi_zero, pi_one, ... pi_B]
  x := linsolve(P, v);
  pi_zero := x[1];

end:
```

The following function `LossProb` is used to compute the probability that a packet is lost in a finite buffer leaky bucket given the size of the packet buffer, the size of the token pool, and the normalized arrival rate. This function makes use of the `DM1K_empty` function listed above.

```
LossProb := proc(B, K, r)

 local q0, qK, p, T0, f, QK; #variable declarations
 global P;

   # Find the probability of the token pool being empty.
   T0 := DM1K_empty(B, r);

   # P is the approximate equation of the infinite buffer
   # M/G/1 queue's PGF.
   P := (exp(((r*r*exp(-r)-2*r-exp(-r)+1+r*exp(-r))/
        (exp(-r)-1))*(z-1)))*(1-r)*(1-z)/
        ((exp(((r*r*exp(-r)-2*r-exp(-r)+1+r*exp(-r))/
        (exp(-r)-1))*(z-1)))-z);

   p := proc(x)  # The function p(x) finds the probability that
                 # the M/G/1 queue contains x pakets.
     if x = 0 then
       evalf(limit(P, z=0));
     else
       evalf(limit(diff(P, z$x), z=0)/x!);
     fi;
   end;

   # From the results of the M/G/1 queue find q0 and qK --
   # the probability of the M/G/1/K queue being empty and full
   # respectively.
   f := 1/p(0)+r*sum('p(x)','x'=0..K-1);
   qK := 1-((1-f*p(0))/r);
   q0 := f*p(0);

   # Find QK -- the loss probability at the leaky bucket due
   # to a full packet queue.
   QK := (1-(q0/(q0+T0-q0*T0))+((q0*T0)/(q0+T0-q0*T0)))*qK;

   end:
```

I.3 Comments on the Execution Time

The Maple V functions, `leaky_bucket` and `LossProb`, were run on a Sun SPARCstation™ 5 workstation using an 85 MHz microSPARC-II microprocessor, 256 Mbytes of RAM and 16 Kbytes of cache. The execution time, needed to find a single data point with each function ranged between about 2 seconds and about 10 seconds for a token pool of size $B = 2$ and $B = 12$ respectively. The execution time is expected to increase logarithmically as B increases since the size of the transition matrix is related to B and so is the size of the set of linear equations to be solved. The value of K in the `LossProb` function is not as critical as B since it only add a few extra steps to find the probability that the M/G/1 queue is full and, therefore, does not increase the execution time dramatically.

The Maple V built in function `plot` can be used to plot the mean queue length or the loss probability, with `leaky_bucket` or `LossProb` respectively, for a range of values of ρ at a surprising speed. This is due to the way Maple V works since it performs all the computations through symbol manipulation. The evaluation of the results occurs only at the end. This causes the function to be executed once and the `plot` function to evaluate the results quickly for the different values of ρ at the end.

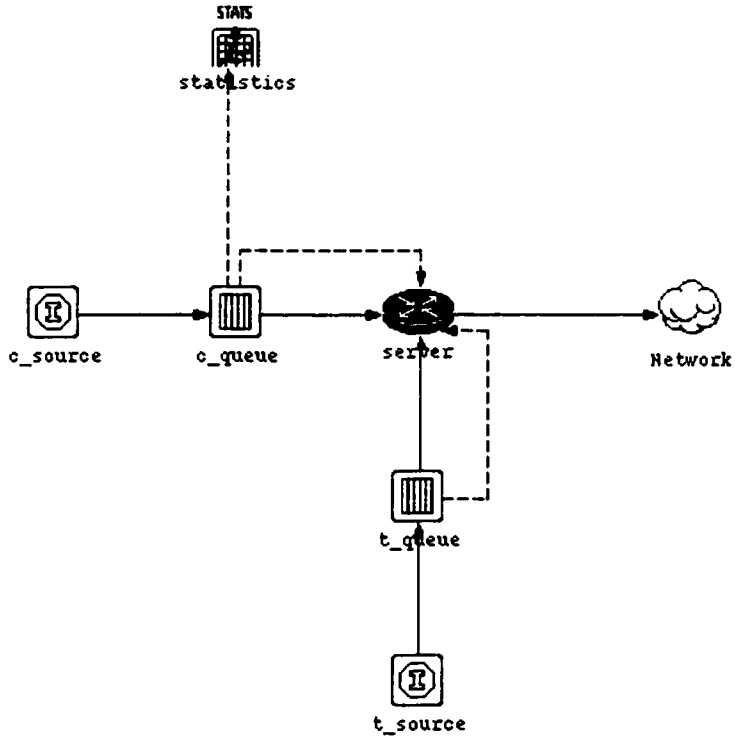
In comparison, the execution time of the simulation run is not related to the size of the token pool nor to the size of the packet buffer. The value of ρ will have an effect on the simulation time since it controls the number of generated packets. The simulation time required to find a single data point, however, is about 10,000,000 seconds for the mean queue length results to converge, provided that a token and a packet are generated every second (i.e. $\rho = 1$). This translates into 6500 seconds of real CPU time on a Sun Ultra™ 5 Workstation using a 270 MHz UltraSPARC-IIi microprocessor, 128 Mbytes of RAM and 256 Kbytes of cache. For lower values of ρ the required simulation time could be reduced relatively to ρ. In addition, multiple runs are required if a confidence interval is to be

found. The required simulation time for the packet loss probability to converge must be at least 10 times longer.

Appendix II: OPNET 4.0 Simulation Model

II.1 Node Level Model

The node level of the model is presented in the following graph.



The node report is presented in the eight pages that follow. This node report describes the leaky bucket model with an infinite packet buffer. The model can be, easily, converted to a finite size leaky bucket by limiting the size of the packet buffer. This can achieved by setting the `pk capacity` parameter of the `c_queue.subqueue[0]` to `promoted` instead of `infinity`. This promotion will allow the queue size to be assigned at simulation time through the attribute `c_queue.subqueue[0].pk capacity`. The token pool, `t_queue`, can be used as an example.

...
...

Node Model Comments

Node Model Types

Node Type	*Supported*	*Default Icon*
fixed	YES	fixed_comm
mobile	YES	mobile_comm
satellite	YES	sat_comm

Node Model Interface Attributes

Attribute **altitude** properties

Property	*Value*	*Inherit*
Assign Status:	set	
Initial Value:	0.0	N/A
Default Value:	0.0	YES
Data Type:	double	N/A
Attribute Description:	Private	N/A
Comments:		YES
	This attribute represents the distance of a satellite from sea-level on the surface of the earth.	
Symbol Map:	NONE	YES

Attribute **c_source.interarrival args** properties

Property	*Value*	*Inherit*
Assign Status:	promoted	
Default Value:	1.0	YES
Data Type:	string list	N/A
Attribute Description:	Private	N/A
Comments:		YES
	This attribute is a list of numerical parameters which complete the specification for computation of packet interarrival time.	
Symbol Map:	NONE	YES

Attribute **condition** properties

Property	*Value*	*Inherit*
Assign Status:	set	
Initial Value:	enabled	N/A
Default Value:	enabled	YES
Data Type:	toggle	N/A
Attribute Description:	Private	N/A
Comments:		YES
	This attribute represents the operational status of a node	
Symbol Map:	NONE	YES

...
...

Attribute **phase** properties

Property	*Value*	*Inherit*
Assign Status:	set	
Initial Value:	0.0	N/A
Default Value:	0.0	YES
Data Type:	double	N/A
Attribute Description:	Private	N/A
Units:	sec.	YES
Comments:	This attribute supports the translation in time of a satellite's orbit.	YES
Symbol Map:	NONE	YES

Attribute **priority** properties

Property	*Value*	*Inherit*
Assign Status:	set	
Initial Value:	0	N/A
Default Value:	0	YES
Data Type:	integer	N/A
Attribute Description:	Private	N/A
Low Range:	0 inclusive	YES
High Range:	32000 inclusive	YES
Comments:	This attribute represents a node's priority. When multiple events are scheduled for the same simulation time, those for a higher priority node wiil be executed first.	YES
Symbol Map:	NONE	YES

Attribute **server.Service Time** properties

Property	*Value*	*Inherit*
Assign Status:	promoted	
Default Value:	0.0	YES
Data Type:	double	N/A
Attribute Description:	Private	N/A
Units:	sec	YES
Symbol Map:	NONE	YES

Attribute **t_queue.subqueue [0].pk capacity** properties

Property	*Value*	*Inherit*
Assign Status:	promoted	
Default Value:	infinity	YES
Data Type:	double	N/A
Attribute Description:	Private	N/A
Units:	pks	YES
Low Range:	0.0 inclusive	YES
Comments:	This attribute specifies the maximum number of packets that can be held in the subqueue.	YES
Symbol Map:	NONE	YES

...
...

Attribute **user id** properties

Property	*Value*	*Inherit*
Assign Status:	set	
Initial Value:	0	N/A
Default Value:	0	YES
Data Type:	integer	N/A
Attribute Description:	Private	N/A
Comments:		YES
	This attribute is an integer which is provided for user-specific identification purposes.	
Symbol Map:	NONE	YES

ideal generator **c_source**

attribute	*value*	*type*	*default value*
name	c_source	string	g
interarrival pdf	exponential	typed file	constant
interarrival args	promoted	string list	1.0
pk size pdf	constant	typed file	constant
pk size args	1	string list	1024.0
field (0) pdf	constant	typed file	constant
field (0) args	0.0	string list	0.0
packet format	NONE	typed file	NONE
start time	0.0	double	0.0
stop time	infinity	double	infinity
icon name	ideal_gen	icon	ideal_gen

packet stream **c_source [0] -> c_queue [0]**

attribute	*value*	*type*	*default value*
name	strm_0	string	strm
src stream	src stream [0]	enumerated	src stream [0]
dest stream	dest stream [0]	enumerated	dest stream [0]
intrpt method	scheduled	integer	scheduled
delay	0.0	double	0.0
color	RGB030	color	RGB030

ideal generator **t_source**

attribute	*value*	*type*	*default value*
name	t_source	string	g
interarrival pdf	constant	typed file	constant
interarrival args	1.0	string list	1.0
pk size pdf	constant	typed file	constant
pk size args	1	string list	1024.0
field (0) pdf	constant	typed file	constant
field (0) args	0.0	string list	0.0
packet format	NONE	typed file	NONE

start time	0.0	double	0.0
stop time	infinity	double	infinity
icon name	ideal_gen	icon	ideal_gen

packet stream **t_source [0] -> t_queue [0]**			
attribute	*value*	*type*	*default value*
name	strm_2	string	strm
src stream	src stream [0]	enumerated	src stream [0]
dest stream	dest stream [0]	enumerated	dest stream [0]
intrpt method	scheduled	integer	scheduled
delay	0.0	double	0.0
color	RGB030	color	RGB030

queue **c_queue**			
attribute	*value*	*type*	*default value*
name	c_queue	string	p
process model	pc_fifo	typed file	pc_fifo
icon name	queue	icon	queue
begsim intrpt	disabled	toggle	disabled
endsim intrpt	disabled	toggle	disabled
failure intrpts	disabled	enumerated	disabled
intrpt interval	disabled	toggle double	disabled
priority	0	integer	0
recovery intrpts	disabled	enumerated	disabled
subqueue	(See below.)	compound	
super priority	disabled	toggle	disabled

subqueue **c_queue.subqueue**			
attribute	*value*	*type*	*default value*
count	1	integer	0
list	(See below.)	object list	

list **c_queue.subqueue [0]**			
attribute	*value*	*type*	*default value*
bit capacity	infinity	double	infinity
pk capacity	infinity	double	infinity

packet stream **c_queue [0] -> server [0]**			
attribute	*value*	*type*	*default value*
name	strm_1	string	strm
src stream	src stream [0]	enumerated	src stream [0]
dest stream	dest stream [0]	enumerated	dest stream [0]

intrpt method	scheduled	integer	scheduled
delay	0.0	double	0.0
color	RGB030	color	RGB030

statistic wire **c_queue [pksize] -> server [instat [0]]**			
attribute	*value*	*type*	*default value*
name	stat_0	string	stat
submodule	NONE	enumerated	NONE
src stat	pksize	enumerated	
dest stat	instat [0]	enumerated	instat [0]
intrpt method	scheduled	integer	scheduled
delay	0.0	double	0.0
rising edge trigger	enabled	toggle	enabled
falling edge trigger	disabled	toggle	enabled
repeated value trigger	disabled	toggle	disabled
zero crossing trigger	disabled	toggle	disabled
low threshold trigger	disabled	toggle double	disabled
high threshold trigger	disabled	toggle double	disabled
color	RGB331	color	RGB331

statistic wire **c_queue [pksize] -> statistics [instat [0]]**			
attribute	*value*	*type*	*default value*
name	stat_2	string	stat
submodule	NONE	enumerated	NONE
src stat	pksize	enumerated	
dest stat	instat [0]	enumerated	instat [0]
intrpt method	scheduled	integer	scheduled
delay	0.0	double	0.0
rising edge trigger	enabled	toggle	enabled
falling edge trigger	enabled	toggle	enabled
repeated value trigger	disabled	toggle	disabled
zero crossing trigger	disabled	toggle	disabled
low threshold trigger	disabled	toggle double	disabled
high threshold trigger	disabled	toggle double	disabled
color	RGB331	color	RGB331

queue **t_queue**			
attribute	*value*	*type*	*default value*
name	t_queue	string	p
process model	pc_fifo	typed file	pc_fifo
icon name	queue	icon	queue
begsim intrpt	disabled	toggle	disabled
endsim intrpt	disabled	toggle	disabled
failure intrpts	disabled	enumerated	disabled
intrpt interval	disabled	toggle double	disabled
priority	0	integer	0

...
...

recovery intrpts	disabled	enumerated	disabled
subqueue	(See below.)	compound	
super priority	disabled	toggle	disabled
subqueue [0].pk capacity	promoted	double	infinity

subqueue **t_queue.subqueue**

attribute	*value*	*type*	*default value*
count	1	integer	0
list	(See below.)	object list	

list **t_queue.subqueue [0]**

attribute	*value*	*type*	*default value*
bit capacity	infinity	double	infinity
pk capacity	promoted	double	infinity

packet stream **t_queue [0] -> server [1]**

attribute	*value*	*type*	*default value*
name	strm_3	string	strm
src stream	src stream [0]	enumerated	src stream [0]
dest stream	dest stream [1]	enumerated	dest stream [0]
intrpt method	scheduled	integer	scheduled
delay	0.0	double	0.0
color	RGB030	color	RGB030

statistic wire **t_queue [pksize] -> server [instat [1]]**

attribute	*value*	*type*	*default value*
name	stat_1	string	stat
submodule	NONE	enumerated	NONE
src stat	pksize	enumerated	
dest stat	instat [1]	enumerated	instat [0]
intrpt method	scheduled	integer	scheduled
delay	0.0	double	0.0
rising edge trigger	enabled	toggle	enabled
falling edge trigger	disabled	toggle	enabled
repeated value trigger	disabled	toggle	disabled
zero crossing trigger	disabled	toggle	disabled
low threshold trigger	disabled	toggle double	disabled
high threshold trigger	disabled	toggle double	disabled
color	RGB331	color	RGB331

...
...

processor **server**

attribute	*value*	*type*	*default value*
name	server	string	p
process model	leaky_bucket	typed file	sink
icon name	router_icon	icon	processor
Service Time	promoted	double	0.0
begsim intrpt	enabled	toggle	disabled
endsim intrpt	disabled	toggle	disabled
failure intrpts	disabled	enumerated	disabled
intrpt interval	disabled	toggle double	disabled
priority	0	integer	0
recovery intrpts	disabled	enumerated	disabled
super priority	disabled	toggle	disabled

packet stream **server [0] -> Network [0]**

attribute	*value*	*type*	*default value*
name	strm_4	string	strm
src stream	src stream [0]	enumerated	src stream [0]
dest stream	dest stream [0]	enumerated	dest stream [0]
intrpt method	scheduled	integer	scheduled
delay	0.0	double	0.0
color	RGB030	color	RGB030

processor **statistics**

attribute	*value*	*type*	*default value*
name	statistics	string	p
process model	leaky_stat	typed file	sink
icon name	proc_locstat_prom	icon	processor
begsim intrpt	disabled	toggle	disabled
endsim intrpt	disabled	toggle	disabled
failure intrpts	disabled	enumerated	disabled
intrpt interval	disabled	toggle double	disabled
priority	0	integer	0
recovery intrpts	disabled	enumerated	disabled
super priority	disabled	toggle	disabled

processor **Network**

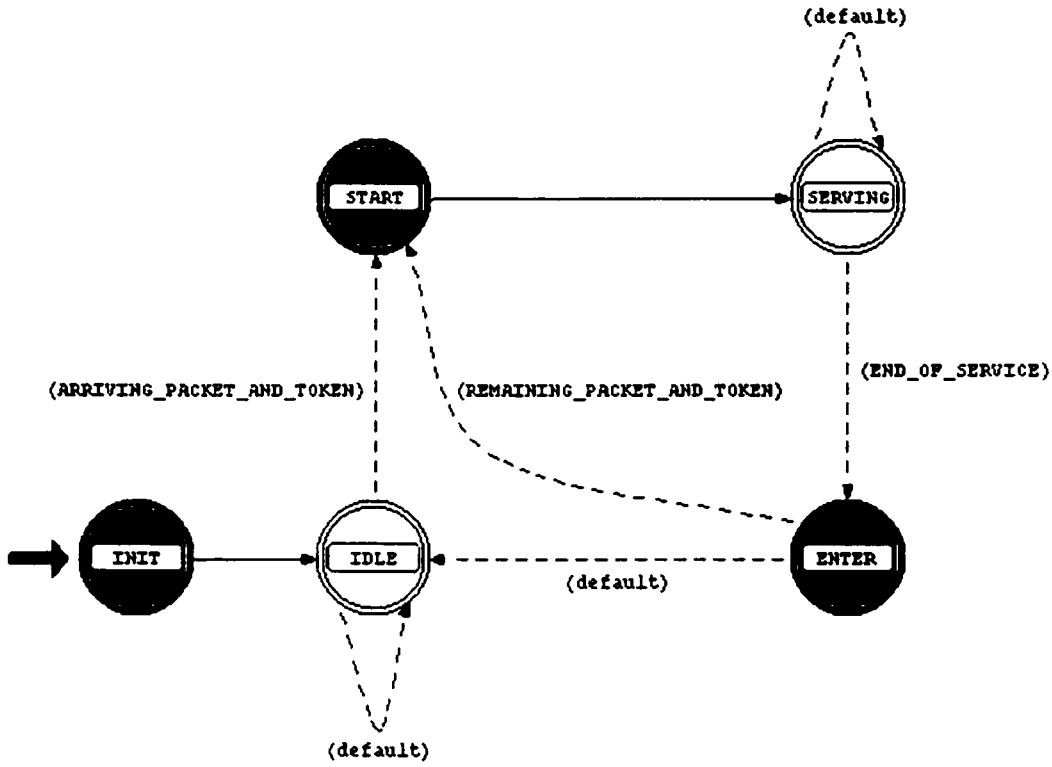
attribute	*value*	*type*	*default value*
name	Network	string	p
process model	sink	typed file	sink
icon name	cloud	icon	processor
begsim intrpt	disabled	toggle	disabled
endsim intrpt	disabled	toggle	disabled
failure intrpts	disabled	enumerated	disabled
intrpt interval	disabled	toggle double	disabled

...
...

priority	0	integer	0
recovery intrpts	disabled	enumerated	disabled
super priority	disabled	toggle	disabled

II.2 The Server Process

The server process is based on the following state diagram.



The detailed report of the server process (called leaky_bucket) is presented in the six pages that follow.

...
...

Process Model Attributes

Attribute **Service Time** properties

Property	*Value*
Default Value:	0.0
Data Type:	double
Attribute Description:	Private
Auto. assign value:	FALSE
Units:	sec

Process Model Interface Attributes

Attribute **begsim intrpt** properties

Property	*Value*	*Inherit*
Assign Status:	set	
Initial Value:	disabled	N/A
Default Value:	disabled	YES
Data Type:	toggle	N/A
Attribute Description:	Private	N/A
Comments:		YES
	This attribute specifies whether a 'begin simulation interrupt' is generated for a processor module's root process at the start of the simulation.	
Symbol Map:	NONE	YES

Attribute **endsim intrpt** properties

Property	*Value*	*Inherit*
Assign Status:	set	
Initial Value:	disabled	N/A
Default Value:	disabled	YES
Data Type:	toggle	N/A
Attribute Description:	Private	N/A
Comments:		YES
	This attribute specifies whether an 'end simulation interrupt' is generated for a processor module's root process at the end of the simulation.	
Symbol Map:	NONE	YES

Attribute **failure intrpts** properties

Property	*Value*	*Inherit*
Assign Status:	set	
Initial Value:	disabled	N/A
Default Value:	disabled	YES
Data Type:	enumerated	N/A
Attribute Description:	Private	N/A
Comments:		YES
	This attribute specifies whether failure interrupts are generated for a processor module's root process upon failure of nodes or links in the network model.	
Symbol Map:	NONE	YES

...
...

Attribute **intrpt interval** properties		
Property	*Value*	*Inherit*
Assign Status:	set	
Initial Value:	disabled	N/A
Default Value:	disabled	YES
Data Type:	toggle double	N/A
Attribute Description:	Private	N/A
Units:	sec.	YES
Comments:		YES
	This attribute specifies how often regular interrupts are scheduled for the root process of a processor module.	
Symbol Map:	NONE	YES

Attribute **priority** properties		
Property	*Value*	*Inherit*
Assign Status:	set	
Initial Value:	0	N/A
Default Value:	0	YES
Data Type:	integer	N/A
Attribute Description:	Private	N/A
Low Range:	-32767 inclusive	YES
High Range:	32767 inclusive	YES
Comments:		YES
	This attribute is used to determine the execution order of events that are scheduled to occur at the same simulation time.	
Symbol Map:	NONE	YES

Attribute **recovery intrpts** properties		
Property	*Value*	*Inherit*
Assign Status:	set	
Initial Value:	disabled	N/A
Default Value:	disabled	YES
Data Type:	enumerated	N/A
Attribute Description:	Private	N/A
Comments:		YES
	This attribute specifies whether recovery interrupts are scheduled for the processor module's root process upon recovery of nodes or links in the network model.	
Symbol Map:	NONE	YES

Attribute **subqueue** properties		
Property	*Value*	*Inherit*
Assign Status:	set	
Initial Value:	(...)	N/A
Default Value:		YES
Data Type:	compound	N/A
Attribute Description:	Private	N/A
Comments:		YES
	This operation attribute permits the addition and	

	deletion of subqueues within the queue module.	
Symbol Map:	NONE	YES

Attribute **super priority** properties

Property	*Value*	*Inherit*
Assign Status:	set	
Initial Value:	disabled	N/A
Default Value:	disabled	YES
Data Type:	toggle	N/A
Attribute Description:	Private	N/A
Comments:		YES
	This attribute is used to determine the execution order of events that are scheduled to occur at the same simulation time.	
Symbol Map:	NONE	YES

Header Block

```
#define PACKET 0  /* Packets enter the server from input stream 0 */
#define TOKEN 1   /* Tokens enter the server from input stream 1 */
#define NETWORK 0 /* Packets are forwared to the network through output stream 0 */
/* ARRIVING_PACKET_AND_TOKEN condition is true when both queues are not empty */
#define ARRIVING_PACKET_AND_TOKEN  ((op_stat_local_read(PACKET) != 0) && (op_stat_local_read(TOKEN) != 0))
/* REMAINING_PACKET_AND_TOKEN condition is similar but it is checked before the counters are decremented */
#define REMAINING_PACKET_AND_TOKEN ((op_stat_local_read(PACKET) > 1) && (op_stat_local_read(TOKEN) > 1))
/* The service time delay is ended when the self-interrupt timer expires */
#define END_OF_SERVICE (op_intrpt_type() == OPC_INTRPT_SELF)
```

State Variable Block

```
/* Define the service_time variable */
double \service_time;
```

unforced state **IDLE**

attribute	*value*	*type*	*default value*
name	IDLE	string	st
enter execs	(empty)	textlist	(empty)
exit execs	(empty)	textlist	(empty)
status	unforced	toggle	unforced

transition **IDLE -> START**

attribute	*value*	*type*	*default value*
name	tr_15	string	tr
condition	ARRIVING_PACKET_AND...	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

...
...

transition **IDLE -> IDLE**

attribute	*value*	*type*	*default value*
name	tr_21	string	tr
condition	default	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **ENTER**

attribute	*value*	*type*	*default value*
name	ENTER	string	st
enter execs	(See below.)	textlist	
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **ENTER**

```
    /* Get token from the token queue */
    op_strm_access(TOKEN);
    /* Get packet from the packet queue */
    op_strm_access(PACKET);
5   /* Destroy token after getting it from the token stream*/
    op_pk_destroy(op_pk_get(TOKEN));
    /* Forward packet to the network after getting it from the packet stream */
    op_pk_send(op_pk_get(PACKET), NETWORK);
```

transition **ENTER -> IDLE**

attribute	*value*	*type*	*default value*
name	tr_5	string	tr
condition	default	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **ENTER -> START**

attribute	*value*	*type*	*default value*
name	tr_9	string	tr
condition	REMAINING_PACKET_AN...	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

...
...

unforced state **SERVING**

attribute	*value*	*type*	*default value*
name	SERVING	string	st
enter execs	(See below.)	textlist	
exit execs	(empty)	textlist	(empty)
status	unforced	toggle	unforced

enter execs **SERVING**

```
```

transition **SERVING -> ENTER**

attribute	*value*	*type*	*default value*
name	tr_17	string	tr
condition	END_OF_SERVICE	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **SERVING -> SERVING**

attribute	*value*	*type*	*default value*
name	tr_25	string	tr
condition	default	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **INIT**

attribute	*value*	*type*	*default value*
name	INIT	string	st
enter execs	(See below.)	textlist	
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **INIT**

```
/* Get the service time attribute assigned at simulation time */
op_ima_obj_attr_get(op_id_self(), "Service Time", &service_time);
```

transition **INIT -> IDLE**

attribute	*value*	*type*	*default value*
name	tr_19	string	tr
condition		string	
executive		string	

...
...

color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **START**

attribute	*value*	*type*	*default value*
name	START	string	st
enter execs	(See below.)	textlist	
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **START**

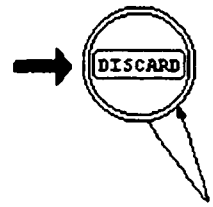
```
/* Schedule a self-interrupt for a duration specified by "service_time" */
op_intrpt_schedule_self(op_sim_time()+service_time, 0);
```

transition **START -> SERVING**

attribute	*value*	*type*	*default value*
name	tr_23	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

II.3 The Network Process

The network process is based on the following state diagram.



The network (or sink) process detailed report can be found in the next four pages.

...
...

Process Model Comments

General Process Description:

The sink process model accepts packets from any number of sources and discards them regardless of their content or format.

ICI Interfaces:

None

Packet Formats:

None

Statistic Wires:

None

Process Registry:

Not Applicable

Restrictions:

None

Process Model Interface Attributes

Attribute **begsim intrpt** properties

Property	*Value*	*Inherit*
Assign Status:	hidden	
Initial Value:	disabled	N/A
Default Value:	disabled	YES
Data Type:	toggle	N/A
Attribute Description:	Private	N/A
Comments:		YES
	This attribute specifies whether a 'begin simulation interrupt' is generated for a processor module's root process at the start of the simulation.	
Symbol Map:	NONE	YES

Attribute **endsim intrpt** properties

Property	*Value*	*Inherit*
Assign Status:	hidden	
Initial Value:	disabled	N/A
Default Value:	disabled	YES
Data Type:	toggle	N/A
Attribute Description:	Private	N/A
Comments:		YES
	This attribute specifies whether an 'end simulation	

...
...

	interrupt' is generated for a processor module's root process at the end of the simulation.	
Symbol Map:	NONE	YES

Attribute **failure intrpts** properties		
Property	*Value*	*Inherit*
Assign Status:	hidden	
Initial Value:	disabled	N/A
Default Value:	disabled	YES
Data Type:	enumerated	N/A
Attribute Description:	Private	N/A
Comments:		YES
	This attribute specifies whether failure interrupts are generated for a processor module's root process upon failure of nodes or links in the network model.	
Symbol Map:	NONE	YES

Attribute **intrpt interval** properties		
Property	*Value*	*Inherit*
Assign Status:	hidden	
Initial Value:	disabled	N/A
Default Value:	disabled	YES
Data Type:	toggle double	N/A
Attribute Description:	Private	N/A
Units:	sec.	YES
Comments:		YES
	This attribute specifies how often regular interrupts are scheduled for the root process of a processor module.	
Symbol Map:	NONE	YES

Attribute **priority** properties		
Property	*Value*	*Inherit*
Assign Status:	hidden	
Initial Value:	0	N/A
Default Value:	0	YES
Data Type:	integer	N/A
Attribute Description:	Private	N/A
Low Range:	-32767 inclusive	YES
High Range:	32767 inclusive	YES
Comments:		YES
	This attribute is used to determine the execution order of events that are scheduled to occur at the same simulation time.	
Symbol Map:	NONE	YES

Attribute **recovery intrpts** properties		
Property	*Value*	*Inherit*
Assign Status:	hidden	
Initial Value:	disabled	N/A
Default Value:	disabled	YES

...
...

Data Type:	enumerated	N/A
Attribute Description:	Private	N/A
Comments:		YES
	This attribute specifies whether recovery interrupts are scheduled for the processor module's root process upon recovery of nodes or links in the network model.	
Symbol Map:	NONE	YES

Attribute **subqueue** properties

Property	*Value*	*Inherit*
Assign Status:	set	
Initial Value:	(...)	N/A
Default Value:		YES
Data Type:	compound	N/A
Attribute Description:	Private	N/A
Comments:		YES
	This operation attribute permits the addition and deletion of subqueues within the queue module.	
Symbol Map:	NONE	YES

Attribute **super priority** properties

Property	*Value*	*Inherit*
Assign Status:	hidden	
Initial Value:	disabled	N/A
Default Value:	disabled	YES
Data Type:	toggle	N/A
Attribute Description:	Private	N/A
Comments:		YES
	This attribute is used to determine the execution order of events that are scheduled to occur at the same simulation time.	
Symbol Map:	NONE	YES

unforced state **DISCARD**

attribute	*value*	*type*	*default value*
name	DISCARD	string	st
enter execs	(See below.)	textlist	
exit execs	(empty)	textlist	(empty)
status	unforced	toggle	unforced

enter execs **DISCARD**

```
/* Destroy packet */
op_pk_destroy (op_pk_get (op_intrpt_strm ()));
```

...
...

transition **DISCARD -> DISCARD**			
attribute	*value*	*type*	*default value*
name		string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

II.4 The Statistics Process

The statistics process is based on the following state diagram.

The statistics process (called leaky_stat) is described in detail in the following five pages.

...
...

Process Model Comments

General Process Description:

The sink process model accepts packets from any number of sources and discards them regardless of their content or format.

ICI Interfaces:

None

Packet Formats:

None

Statistic Wires:

None

Process Registry:

Not Applicable

Restrictions:

None

Process Model Interface Attributes

Attribute **begsim intrpt** properties

Property	*Value*	*Inherit*
Assign Status:	hidden	
Initial Value:	disabled	N/A
Default Value:	disabled	YES
Data Type:	toggle	N/A
Attribute Description:	Private	N/A
Comments:		YES
	This attribute specifies whether a 'begin simulation interrupt' is generated for a processor module's root process at the start of the simulation.	
Symbol Map:	NONE	YES

Attribute **endsim intrpt** properties

Property	*Value*	*Inherit*
Assign Status:	hidden	
Initial Value:	disabled	N/A
Default Value:	disabled	YES
Data Type:	toggle	N/A
Attribute Description:	Private	N/A
Comments:		YES
	This attribute specifies whether an 'end simulation	

	interrupt' is generated for a processor module's root process at the end of the simulation.	
Symbol Map:	NONE	YES

Attribute **failure intrpts** properties		
Property	*Value*	*Inherit*
Assign Status:	hidden	
Initial Value:	disabled	N/A
Default Value:	disabled	YES
Data Type:	enumerated	N/A
Attribute Description:	Private	N/A
Comments:		YES
	This attribute specifies whether failure interrupts are generated for a processor module's root process upon failure of nodes or links in the network model.	
Symbol Map:	NONE	YES

Attribute **intrpt interval** properties		
Property	*Value*	*Inherit*
Assign Status:	hidden	
Initial Value:	disabled	N/A
Default Value:	disabled	YES
Data Type:	toggle double	N/A
Attribute Description:	Private	N/A
Units:	sec.	YES
Comments:		YES
	This attribute specifies how often regular interrupts are scheduled for the root process of a processor module.	
Symbol Map:	NONE	YES

Attribute **priority** properties		
Property	*Value*	*Inherit*
Assign Status:	hidden	
Initial Value:	0	N/A
Default Value:	0	YES
Data Type:	integer	N/A
Attribute Description:	Private	N/A
Low Range:	-32767 inclusive	YES
High Range:	32767 inclusive	YES
Comments:		YES
	This attribute is used to determine the execution order of events that are scheduled to occur at the same simulation time.	
Symbol Map:	NONE	YES

Attribute **recovery intrpts** properties		
Property	*Value*	*Inherit*
Assign Status:	hidden	
Initial Value:	disabled	N/A
Default Value:	disabled	YES

...
...

Data Type:	enumerated	N/A
Attribute Description:	Private	N/A
Comments:		YES
	This attribute specifies whether recovery interrupts are scheduled for the processor module's root process upon recovery of nodes or links in the network model.	
Symbol Map:	NONE	YES

Attribute **subqueue** properties

Property	*Value*	*Inherit*
Assign Status:	set	
Initial Value:	(...)	N/A
Default Value:		YES
Data Type:	compound	N/A
Attribute Description:	Private	N/A
Comments:		YES
	This operation attribute permits the addition and deletion of subqueues within the queue module.	
Symbol Map:	NONE	YES

Attribute **super priority** properties

Property	*Value*	*Inherit*
Assign Status:	hidden	
Initial Value:	disabled	N/A
Default Value:	disabled	YES
Data Type:	toggle	N/A
Attribute Description:	Private	N/A
Comments:		YES
	This attribute is used to determine the execution order of events that are scheduled to occur at the same simulation time.	
Symbol Map:	NONE	YES

Header Block

```
#define PacketQueue 0
```

State Variable Block

```
/* Variables required for calculating the mean queue length */
/* are defined here */
double \CurrentTime;
double \PreviousTime;
double \Total;
double \QueueLength;
double \MeanPacketQueueLength;
Stathandle \MeanQueueHandle;
```

...
...

forced state **INIT**

attribute	*value*	*type*	*default value*
name	INIT	string	st
enter execs	(See below.)	textlist	
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **INIT**

```
   /* Initialize the variables required to calculate the mean queue length */
   Total = 0;
   PreviousTime = 0;
   QueueLength = 0;
5  MeanPacketQueueLength = 0;
   /* Convert the statistic name "Mean Queue (packets)" into a statistics handle */
   MeanQueueHandle = op_stat_reg("Mean Queue (packets)",
                OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
```

transition **INIT -> MEAN**

attribute	*value*	*type*	*default value*
name	tr_12	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

unforced state **MEAN**

attribute	*value*	*type*	*default value*
name	MEAN	string	st
enter execs	(See below.)	textlist	
exit execs	(See below.)	textlist	
status	unforced	toggle	unforced

enter execs **MEAN**

```
   /* THIS FUNCTION CALCULATES THE TIME AVERAGE MEAN QUEUE LENGTH */

   /* Get the current time */
   CurrentTime = op_sim_time();
5
   /* Add the queue length during the last time slot to the */
   /* integral of queue length over the simulation time */
   Total = QueueLength *(CurrentTime - PreviousTime) + Total;

10 /* The mean queue length is the integral of queue length over the */
   /* simulation time divided by the simulation time */
   MeanPacketQueueLength = Total/CurrentTime;

   /* Update the "Mean Queue (packets)" statistic */
```

...
...

```
15  op_stat_write(MeanQueueHandle, MeanPacketQueueLength);

    /* Get the current queue length to be used for the next time slot */
    QueueLength = op_stat_local_read(PacketQueue);

20  /* Debug mode only: Print updated values for debugging purposes */
    if (op_sim_debug() == OPC_TRUE)
     {
     printf("Queue Length = %g \n", QueueLength);
     printf("Mean Queue Length = %g \n", MeanPacketQueueLength);
25   }

    /* Mark the start of the time slot */
    PreviousTime = CurrentTime;
```

exit execs **MEAN**

transition **MEAN -> MEAN**			
attribute	*value*	*type*	*default value*
name	tr_14	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline