Dynamics of an Abstract Overflow Processing Network: The Discrete Case

R. Heuijerjans - 0586875

Rapportnr. SE 420630
Bachelor Final Project
Eindhoven University of Technology

Supervisor:
Prof.dr.ir. J.E. Rooda

Advisors:
Dr.ir. A.A.J. Lefeber
Dr. Y. Nazarathy

Department Mechanical Engineering
Systems Engineering Group

November 30, 2010

Contents

Chapter 1

Introduction

A regular Jackson network is a network of interconnected M/M/1 queuing models, called nodes from now on. An M/M/1 queue is a Kendall's notation for a queue with Poisson process arrival rates and service rates and one workstation with an infinite queue length. In a Jackson network processed jobs can be routed to other nodes or leave the network in a probabilistic manner. Due to the fact that the queue length is infinite the total inflow in a node may not exceed the processing time of this node, otherwise the network is unstable and it will never be in steady state since the number of jobs in the network will increase to infinity. Thus steady state calculations cannot be made. Jackson networks are well studied.

Now the total inflow in one node is allowed to exceed the processing time of this node. To prevent the number of jobs to grow to infinity finite buffer sizes are introduced, creating a network of interconnected M/M/1/K queues. This Jackson Network with overflows is also stable since the finite buffer sizes prevent the number of jobs to grow to infinity. Jobs arriving at a full queue are now be rerouted to another node in a probabilistic way. Only a little is known about this network type of network.

A previous Bachelor Final Project by Stijn Fleuren [2] studies the continuous deterministic model of the Jackson Network with overflows. This can be interpreted as a Jackson Network where all flows in steady state are constant flows of liquid. Therefore the continuous deterministic model of this network is also called the fluid model.

In this Bachelor Final Project an attempt is made to describe the discrete stochastic model of the Jackson Network with overflows. This is done by comparing this scalings of this model to the fluid model. Scalings can be interpreted with the following analogy:

1. One machine 1 processes one lot every hour. This lot is a pallet with 60 boxes.
2. Then this machine is then scaled with factor 60, now we get a machine 2 that processes 1 box every minute. In these boxes there are 60 packets of rice.
3. When scaling machine 2 with factor 60 machine 3 created. Machine 3 produces 1 pack of rice every second.
4. Scaling machine 3 with factor 25000 makes machine 4. This machine 4 produces 1 grain of rice every $\frac{1}{25000}$ second.

The idea behind this scaling is that when scaling is increased the solutions of the scaled discrete stochastic model converges to the solutions of the fluid model. One could imagine that the discrete stochastic model of machine 4 looks more like the fluid model than the discrete stochastic model of machine 1 does.

The actual goal of this project can be expressed by the main question:
Do solutions for a discrete stochastic model of a Jackson Network with overflows converge, after scaling, to solutions for a continuous deterministic model of a Jackson Network with overflows?

The main question is divided in two parts:

1. Does the total inflow in a node in the discrete stochastic model converge after scaling to the inflow in a node in the fluid model?

2. Does the sojourn time distribution of the discrete stochastic model converge after scaling to the sojourn time distribution in the fluid model?

These questions are addressed in this report.

The structure of this report is as follows; first in Chapter 2 the characteristics of the Jackson network with overflows and notations used in this report are explained. In Chapter 3 the models used to address the previously named questions are described. Then Chapter 4 validates two discrete stochastic models. The convergence of the total inflow in a node in the discrete stochastic model to the fluid model is shown in Chapter 5, and convergence of the sojourn time distribution is shown in Chapter 6.

Chapter 2

Network details

The network and its parameters are explained in this chapter.

2.1 Network

In the Jackson network with overflows nodes are numbered $1, 2, \ldots, M$. Each node i processes jobs according a Poisson process with rate μ_i. Jobs also enter the network according a Poisson process in each node with rate α_i. This means production times and inter-arrival times are exponentially distributed with respectively mean $\frac{1}{\mu_i}$ and mean $\frac{1}{\alpha_i}$. Every node has a finite queue size K_i. A two node Jackson network with overflows is shown in Figure 2.1.

When a node i has finished processing a job, this job can move to another node j with probability p_{ij}. Alternatively, the job leaves the system with a probability $1 - \sum_{j=1}^{m} p_{ij} \geq 0$. Furthermore, a node is not allowed to route to itself so $p_{ii} = 0$. All routing probabilities together form a $m \times m$ routing matrix P which has zeros on its diagonal and the row sums are never greater than 1, according to the previously named conditions.

A job can arrive at a full node. This node cannot accept this job and thus the job is rerouted. Like regular routing according to P, rerouting is also probabilistic. The rerouting probabilities are given by the $m \times m$ matrix Q, which has the same characteristics as matrix P.

The total inflow rate in a node i is a sum of α_i and flows rerouted from other nodes to i, this total is named λ_i. The buffer level of node i at time t is called $X_i(t)$.

All rates, network parameters and buffer levels can be represented in vectors and matrices. This is shown in (2.1) and (2.2).

$$\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_m \end{pmatrix}, \quad \alpha = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{pmatrix}, \quad \lambda = \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_m \end{pmatrix}, \quad K = \begin{pmatrix} K_1 \\ K_2 \\ \vdots \\ K_m \end{pmatrix}, \quad X(t) = \begin{pmatrix} X_1(t) \\ X_1(t) \\ \vdots \\ X_m(t) \end{pmatrix}, \tag{2.1}$$

$$P = \begin{pmatrix} 0 & p_{12} & \ldots & p_{1m} \\ p_{21} & 0 & \ldots & p_{2m} \\ \vdots & \vdots & \ddots & \\ p_{m1} & p_{m2} & & 0 \end{pmatrix}, \qquad Q = \begin{pmatrix} 0 & q_{12} & \ldots & q_{1m} \\ q_{21} & 0 & \ldots & q_{2m} \\ \vdots & \vdots & \ddots & \\ q_{m1} & q_{m2} & & 0 \end{pmatrix}. \tag{2.2}$$

.

As explained in the Chapter 1, the discrete stochastic model can be scaled. This is achieved by multiplying the rate-, network parameter-, and buffer level vectors with scale factor N (2.3). From now on, solutions of the fluid model have no superscripts and solutions of the discrete stochastic model have the value of N as superscript.

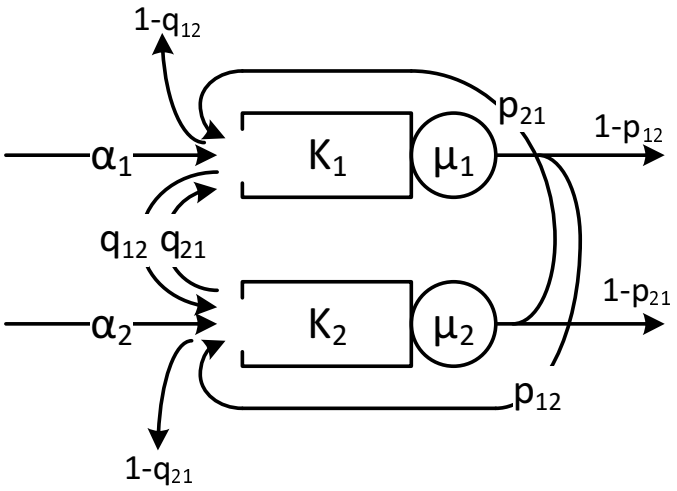


Figure 2.1: A two node Jackson network with overflows. Note that the routing and rerouting flows are represented by probabilities, which are not actual flows.

$$\begin{aligned} \mu^N &= N\mu\,, \\ \alpha^N &= N\alpha\,, \\ K^N &= NK\,, \\ X^N(t) &= X(t)\,. \end{aligned}$$

All network parameters are described in this chapter. Moreover, the difference in notation between solutions for the discrete stochastic model and the solutions for the fluid model are distinguished. In the next chapter the models used in this report are described.

Chapter 3

Modeling

Several different models are used in this report. This chapter aims to discuss and describe these models and name their advantages and disadvantages as well as their range of use. First the χ-model used for simulations is described, then the analytical Markov chain model is explained and finally the fluid model is described.

3.1 χ-Model

The χ-model of the Jackson network is a discrete stochastic model. The χ-specification made for this project is found in Appendix A. Below the χ-model is further explained.

A possible χ model of the Jackson network with overflows can be schematically described by Figure 3.1. The starting process S calls central buffer process B, m instances of generators G and processors M and one exit process E. Bundles in χ normally start at 0, therefore $i = 0, 1, \ldots, (m-1)$ as opposed to $i = 1, \ldots, M$ is used as numbering for the generators, processors and channels.

Process B contains all buffers and also manages the routing of jobs. For this routing a function is used that uses row i of a routing matrix P or Q , depending on if a lot is routed or rerouted, and a sample from a uniform distribution $[0, 1]$ as input. Row i of the routing matrix consists all relevant probabilities p_{ij} or q_{ij}. Then the function determines the destination node j. If the job cannot enter node j because its buffer is full, the function is called again with a new sample from the uniform distribution until a non-full node is found or the outcome of the function is to leave the network.

The advantage of using a χ-model is that large networks can be evaluated and the sojourn time of jobs can be retrieved easily. However, a disadvantage are the long simulation times that are needed before the statistical estimate of a solution is good.

3.2 Markov chain model

The Matlab function for the Markov chain model is found in Appendix B.

Solutions of the discrete stochastic model of the Jackson network with overflows can also be obtained mathematically. Therefore, Markov chain theory is used [3].

Markov chain theory makes use of states and rates for which the current state could move to another. When the network is in steady state the fraction of time the network spends in each state does not change. The steady state distribution π is a vector and contains all fractions of time the system spends in each state. To obtain the steady state distribution for each state a balance equation is formulated. Let r_{ij} be the rate for which the Markov chain moves from state i to j and r_i be the sum of all outgoing rates of this state, then the balance equation for state i would be (3.1).

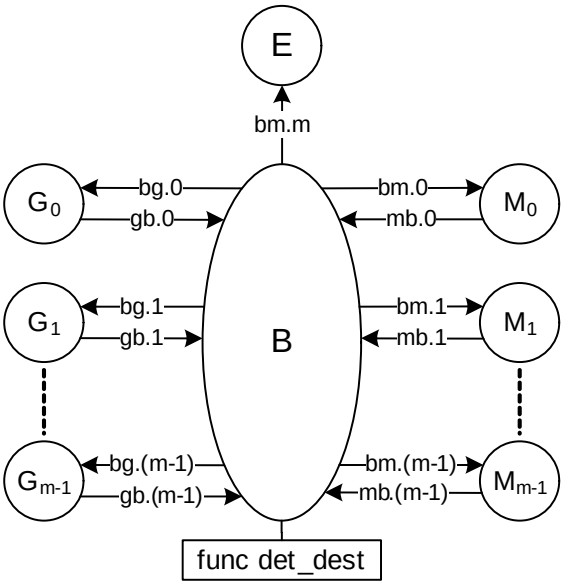


Figure 3.1: Schematic view of the χ-model

$$\pi_i r_i = \sum_j \pi_j r_{ij} \qquad \text{with} \quad r_i = \sum_i r_{ij} \tag{3.1}$$

Defining generator matrix G as (3.2) all balance equations form the system of equations (3.3) and allow one to solve for π.

$$g_{ij} = \begin{cases} -\sum_i r_{ij} & \text{if } i = j; \\ r_{ij} & \text{if } i \neq j. \end{cases} \tag{3.2}$$

$$\pi G = \bar{0} \qquad \text{with} \quad \sum_i \pi_i = 1. \tag{3.3}$$

The Markov chain for the Jackson network with overflows has $\prod_{i=1}^{M}(K_i + 1)$ states, where K_i is the buffer length of node i. Since the size of the system of equations is equal to the number of states, the system of equations could easily become pretty large, for example a Jackson network with overflows with only two nodes each having a buffer length of 50 already lead to a system of $51^2 = 2601$ equations. Therefore only a network for two nodes is evaluated.

First a state space is defined (3.4). In this state space there are 9 sets of states. States belonging to the same set have in common the rates for which the system can move to neighbouring states. The rates depends on whether buffers are full (F), empty (E) or none of both (N). The sets of states are listed in Table 3.1.

$$S = \big\{(X_1, X_2) | X_1 \in \{0, \ldots, K_1\}\ ,\ X_2 \in \{0, \ldots, K_2\}\big\}. \tag{3.4}$$

Then the rates for each set are calculated. These are shown in Figure 3.2.

As the rates are known the generator matrix G can be assembled. In the generator matrix each state has its own row, and because the state space has two dimensions an index function (3.5) is needed that gives each state a unique number so its solution can be stored in a one dimensional vector. Note that the generator matrix is a sparse matrix so efficient solving algorithms can be used, Matlab chooses automatically when matrix G is stored as sparse.

$$\text{index} = f(i, j) = 1 + X_i + X_j(K_1 + 1). \tag{3.5}$$

Set S	Node 1	Node 2
NN	$0 < X_1 < K_1$	$0 < X_2 < K_2$
NE	$0 < X_1 < K_1$	$X_2 = 0$
EN	$X_1 = 0$	$0 < X_2 < K_2$
NF	$0 < X_1 < K_1$	$X_2 = K_2$
FN	$X_1 = K_1$	$0 < X_2 < K_2$
EF	$X_1 = 0$	$X_2 = K_2$
FE	$X_1 = K_1$	$X_2 = 0$
EE	$X_1 = 0$	$X_2 = 0$
FF	$X_1 = K_1$	$X_2 = K_2$

Table 3.1: Sets of states

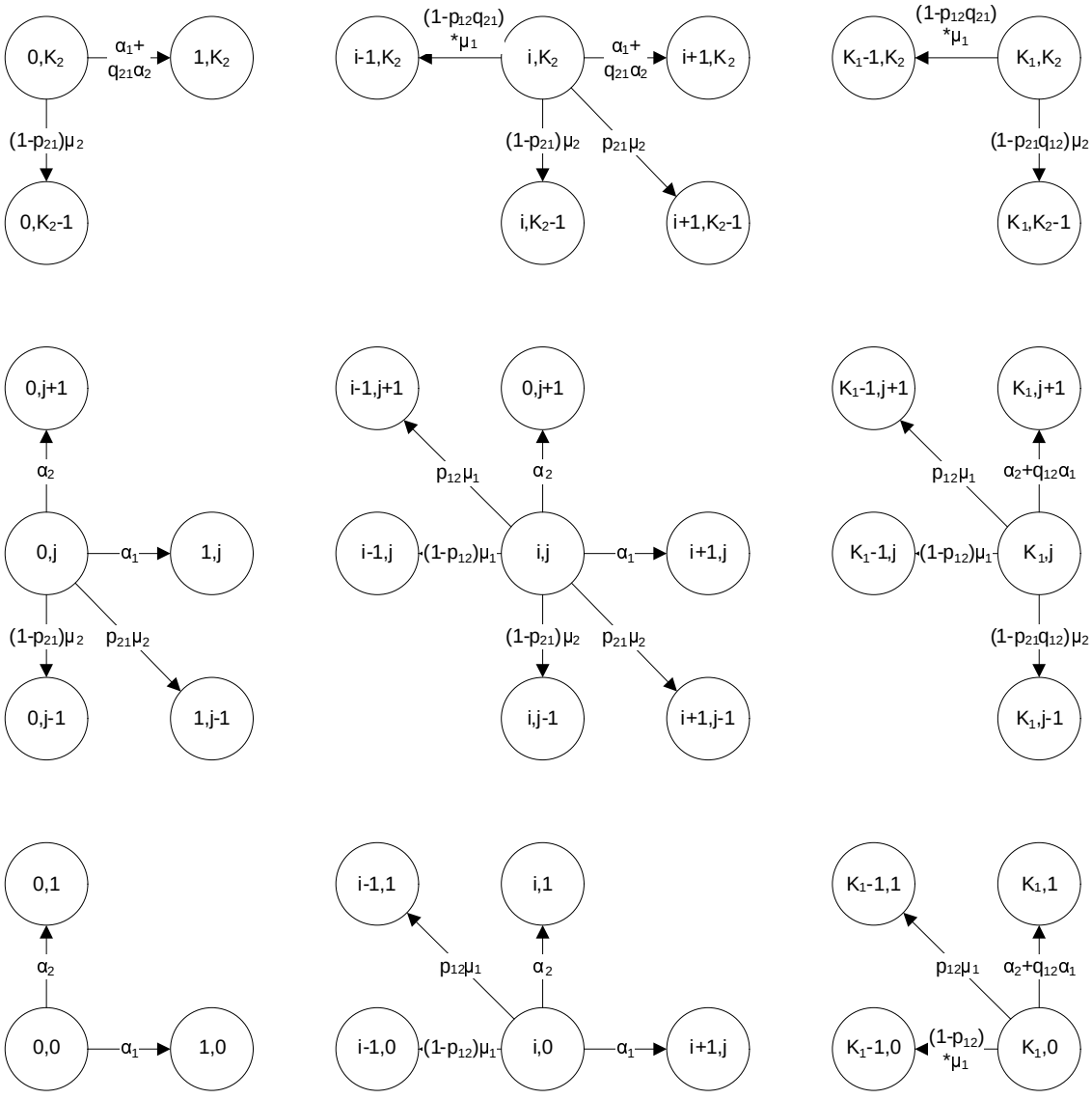


Figure 3.2: State Space with rates of a two node Jackson network with rerouting

1	1-2	1-2-1	1-2-1-2
		1-2-2	1-2-2-1
			1-2-2-2
2	2-2	2-2-1	2-2-1-2
		2-2-2	2-2-2-1
			2-2-2-2
	2-1	2-1-2	2-1-2-1
			2-1-2-2

Table 3.2: Possible routes untill depth 4

The advantage of using the Markov chain model is that solutions are quickly obtained for small networks.

3.3 Fluid model

3.3.1 Total inflows

The fluid model that calculates all total inflows λ_i is a result of Stijn Fleurens Bachelor Final Project [2]. For this project only the transition from Mathematica to a Matlab function is made.

The inflows λ_i for a model of the two node Jackson network with overflows are given in (3.6). The total inflow λ_i depends on whether there is overflowing or not.

$$\begin{array}{lcll} \lambda_1 &=& \alpha_1 + \lambda_2 p_{21} & \text{for} \quad \lambda_2 \leq \mu_2\,, \\ \lambda_2 &=& \alpha_2 + \lambda_1 p_{12} & \text{for} \quad \lambda_1 \leq \mu_1\,, \\ \lambda_1 &=& \alpha_1 + \mu_2 p_{21} + (\lambda_2 - \mu_2) q_{21} & \text{for} \quad \lambda_2 \geq \mu_2\,, \\ \lambda_2 &=& \alpha_2 + \mu_1 p_{12} + (\lambda_1 - \mu_1) q_{12} & \text{for} \quad \lambda_1 \geq \mu_1\,. \end{array} \tag{3.6}$$

3.3.2 Sojourn Time Distribution

For the fluid model of the Jackson Network with overflows a script is made that computes the sojourn time distribution SD for the situation where one buffer is empty and one buffer is full.

For the fluid model of the Jackson Network with overflows the sojourn time distribution is a discrete function. Both sojourn time and fraction of fluid leaving at this sojourn time are a function of the route. When node 1 is full and node 2 is empty, fluid can enter node 2 and cannot enter node 1 again after just being processed by node 1, but fluid just processed by node 2 can enter node 1 and node 2 since it can, but does not have to, be rejected in node 1 and rerouted to node 2. In Table 3.2 all possible routes until depth 4 are given.

The Matlab script in Appendix E uses a recursive function to calculate sojourn time and fraction of fluid leaving for each route until a predefined maximum sojourn time is reached. A recursive function is used because each route depends on a shorter route. The solutions for sojourn time and fraction of fluid leaving for this shorter route are used to calculate these values for the longer route. Let e be the amount of fluid that leaves at time t_e, x the amount of fluid that is just processed by node i where it entered at time t, then the recursive function would look as in (3.7).

$$f\,(x,t,i) = \begin{cases} \begin{array}{rrcl} i=1: & e(k) &=& x(1-p_{12}) \\ & t_e &=& t + \frac{K_1}{\mu_1} \\ \text{call } f(x\,p_{12}, t_e, 2) & & & \\ & & & \\ i=2: & e(k) &=& x\left(1 - p_{21} + p_{21}(1-\frac{\mu_1}{\lambda_1})(1-q_{12})\right) \\ & t(k) &=& t(k_p) + \frac{1}{\mu_2} \\ \text{call } f(x\,(p_{21}\frac{\mu_1}{\lambda_1}), t_e, 1) \text{ and } f(x\,(p_{21}\frac{1-\mu_1}{\lambda_1})\,q_{12}, t_e, 2) & & & \end{array} \end{cases} \tag{3.7}$$

Chapter 4

Comparing the χ-Model and the Markov chain model

In Chapter 3 the models used in this report were discussed. In this chapter the χ-model and the Markov chain model are compared to gain confidence in both models. Once both models are validated they can be used further in this report while having faith in their solutions.

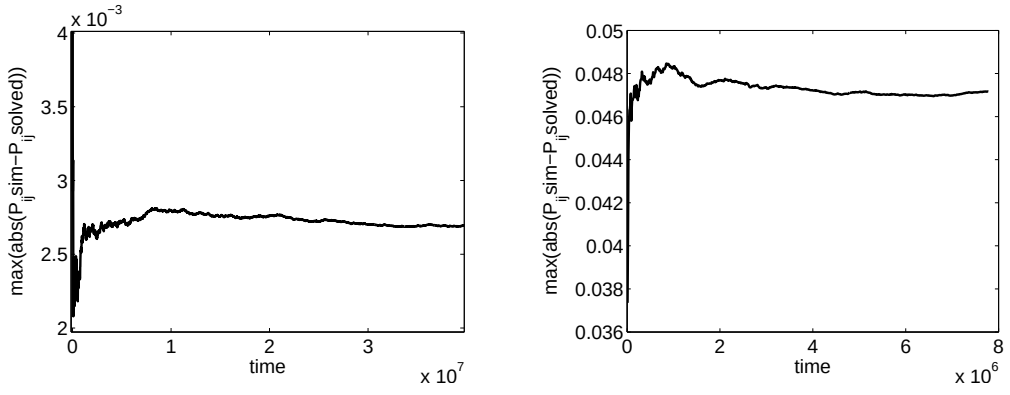
4.1 Approach

When both discrete stochastic models are equal the steady state distribution π^T resulting from the χ-model after runtime T will converge to the Markov chain model steady state distribution π. Since π is a vector the maximum absolute error is used as error measure (4.1). This error measure goes to zero when T increases and both models are equal.

$$\text{error} = \max_i \left(|\pi_i^T - \pi_i| \right) \tag{4.1}$$

4.2 Results

Two different instances of the χ-model are simulated for a long time, generating an output π^T for $T = 10000, 20000, 30000, \ldots$.



(a) Both buffers empty

(b) One buffer empty, one buffer full

Figure 4.1: Error versus simulation time

In Figure 4.2 the error measure of a discrete stochastic Jackson network with overflows with parameters (4.2) is shown for increasing simulation time. For this instance of the network the fluid solution yield that both buffers are empty. It is observed that the error measure converges to $2.5 \cdot 10^{-3}$. This is a quite small error.

Figure 4.3 shows the error measure of a network with parameters (4.2) for an increasing simulation time. For this parameters the fluid solution yields one empty and one full buffer. The error measure converges to about 0.047 which is a quite large error.

$$\mu = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \ \alpha = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix} \ K = \begin{pmatrix} 10 \\ 10 \end{pmatrix} \ P = \begin{pmatrix} 0 & 0.5 \\ 0.0 & 0 \end{pmatrix} \ Q = \begin{pmatrix} 0 & 0.5 \\ 0.5 & 0 \end{pmatrix} \ \lambda = \begin{pmatrix} 0.5 \\ 0.75 \end{pmatrix}. \quad (4.2)$$

$$\mu = \begin{pmatrix} 1 \\ 0.5 \end{pmatrix} \ \alpha = \begin{pmatrix} 0.5 \\ 0.7 \end{pmatrix} \ K = \begin{pmatrix} 10 \\ 10 \end{pmatrix} \ P = \begin{pmatrix} 0 & 0.3 \\ 0.1 & 0 \end{pmatrix} \ Q = \begin{pmatrix} 0 & 0.3 \\ 0.2 & 0 \end{pmatrix} \ \lambda = \begin{pmatrix} 0.6277 \\ 0.8883 \end{pmatrix}. \quad (4.3)$$

Although the error for the network instance with fluid solutions one empty and one full buffer is not satisfying, it is decided to continue. Since there are 121 states a small deviation in one of these states can happen but does not have to influence flows that much.

Chapter 5

Convergence of flowtime

Now that the χ-model and the Markov chain model are validated in the previous chapter, the Markov chain model is used to show that the discrete stochastic solution of total inflow in a node converges to its fluid solution when scaling goes to infinity. Before this is shown, the trajectory of one discrete stochastic M/M/1/K queue for several scalings is compared to its fluid counterpart.

5.1 One node trajectory

Two trajectories for a M/M/1/K queue are simulated for several scalings N trajectories. Both fluid trajectories are taken from [2]. One trajectory is increasing (5.1) with $\alpha_1 > \mu_1$ leading to a full buffer in steady state and the other one (5.2) is decreasing with $\alpha_1 < \mu_1$ and the buffer is empty in steady state. For both the increasing and the decreasing case the trajectories are plotted in respectively Figures 5.1 and 5.2. It is observed that for when scaling is larger, the trajectory of the discrete stochastic model approaches the fluid trajectory.

$$\alpha_1 = 1\,, \qquad \mu_1 = 0.5\,, \qquad K_1 = 10\,, \qquad X_1(0) = 5\,. \tag{5.1}$$

$$\alpha_1 = 0.5\,, \qquad \mu_1 = 1\,, \qquad K_1 = 10\,, \qquad X_1(0) = 5\,. \tag{5.2}$$

5.2 Approach to show convergence of flowtime

One goal of this project is, as mentioned in Chapter 1, to see whether the inflow in a node in the fluid model of a Jackson network with overflows is equal to the inflow in a node in the discrete stochastic model when scaling N goes to infinity. In (5.3) it is mathematically written.

$$\text{Show that } \lambda_i = \lim_{N\to\infty} \lambda_i^N \text{ holds.} \tag{5.3}$$

In other words, when scaling N increases, the lot sizes reduce with factor N while the number of lots, the buffer lengths and rates increase with factor N until the discrete stochastic flow looks like a continuous stream of fluid. The scalings are shown in (5.4).

$$\begin{aligned} \mu^N &= N\mu\,, \\ \alpha^N &= N\alpha\,, \\ K^N &= NK\,. \end{aligned} \tag{5.4}$$

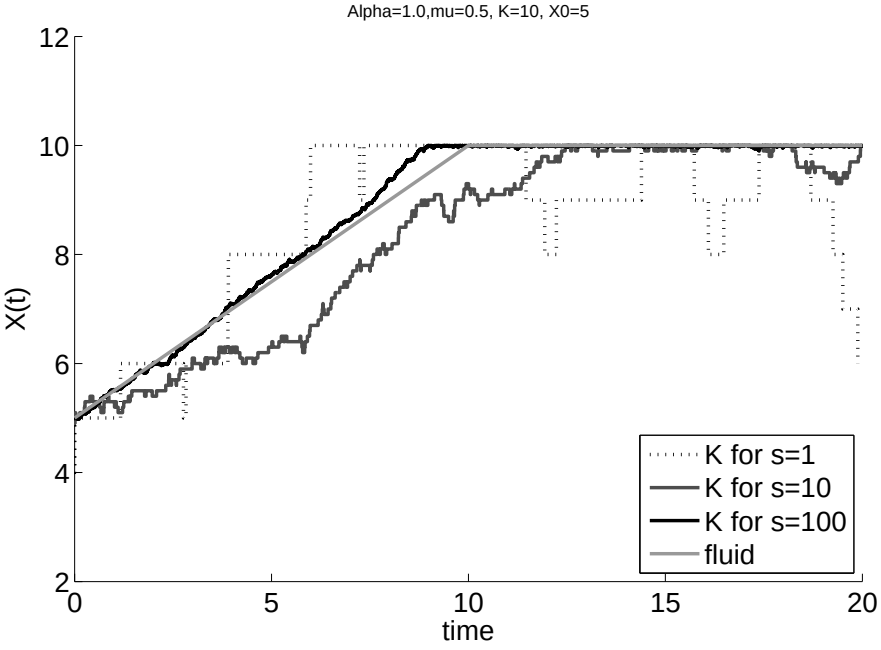


Figure 5.1: Fluid trajectory and stochastic trajectories

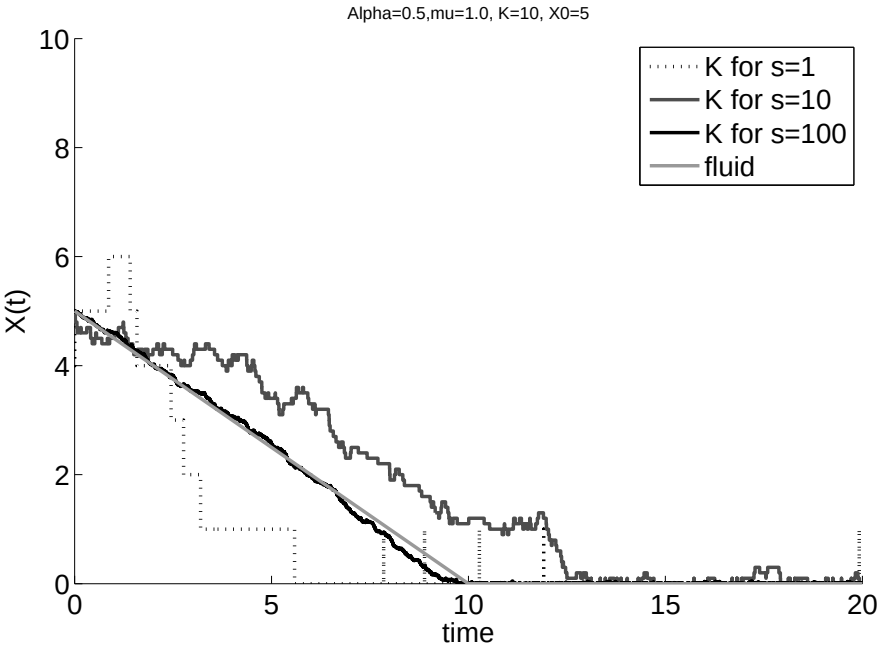


Figure 5.2: Fluid trajectory and stochastic trajectories

5.2.1 Calculating total inflow

The inflows λ_i^N are calculated from the steady state probabilities calculated with the Markov chain model. To calculate λ_i^N the limiting probability for each state is multiplied with the inflow rates that take place in this state. As described in section 3 each state can be categorized in a certain set, thus all time fractions $\pi_{Xi,Xj}$ for states belonging to this set can be added together and multiplied with the according rates (5.5)),

$$\lambda_i^N = \sum_{x=1} \lambda_i^N(x)\pi_x = \sum_{S_x} \lambda_i^N(S_x) \sum_{x \in S_x} \pi_x. \tag{5.5}$$

Inflow $\lambda_i^N(S_x)$ as a function of set of states is determined by the following scheme. Note that all α_i and μ_i used for the calculation of $\lambda_i^N(S_x)$ are unscaled. For node 1 (node 2 is similar):

1. Node 1 receives α_1 (for every set of states)
2. When node 2 is not empty node 1 receives $\mu_2 p_{21}$ due to regular production in node 2.
3. When node 2 is full then node 1 receives $\alpha_2 q_{21}$ (regular rerouting) and $\mu_1 p_{12} q_{21}$ (bounce back)
4. When both nodes are full, point 1, 2 and 3 still hold. However, because both nodes are full any lot entering the system (inputs α_1 and α_2) can be bounced (ping-ponged) between nodes unlimited times. A lot just produced in node 1 does not 'ping-pong' since it left an empty space in node 1 and can enter node 1 after being bounced back once by node 2.

   So the inflow in node 1 due to α_1 becomes as stated in the equation below [1]:

$$\alpha_1(1 + q_{12}q_{21} + q_{12}q_{21}q_{12}q_{21} + \cdots) = \alpha_1 \sum_{n=0}^{\infty} (q_{12}q_{21})^n = \frac{\alpha_1}{1 - q_{12}q_{21}} \tag{5.6}$$

   The inflow due to $\alpha_2 q_{21}$ becomes:

$$\alpha_2 q_{21}(1 + q_{12}q_{21} + q_{12}q_{21}q_{12}q_{21} + \cdots) = \alpha_2 q_{21} \sum_{n=0}^{\infty} (q_{12}q_{21})^n = \frac{\alpha_2 q_{21}}{1 - q_{12}q_{21}} \tag{5.7}$$

In Appendix C a Matlab script for discrete stochastic flow calculation is included.

5.3 Experiments

To show that (5.3) holds the Markov chain model is evaluated for seven different instances of the Jackson network with overflows. For each of the situations one buffer empty and one full, both buffers empty and both buffers full at least two instances are evaluated. Each instance is scaled with factors (5.8).

$$N = 1, 2, \ldots, 24, 25\,. \tag{5.8}$$

The buffer lengths for both nodes at scaling 1 are chosen to be 2 because it is the smallest buffer length that would yield 9 states in the Markov chain, which is the minimum to work properly (Figure 3.1).

5.4 Results

The results are categorized in one buffer empty and one full, both buffers empty and both buffers full.

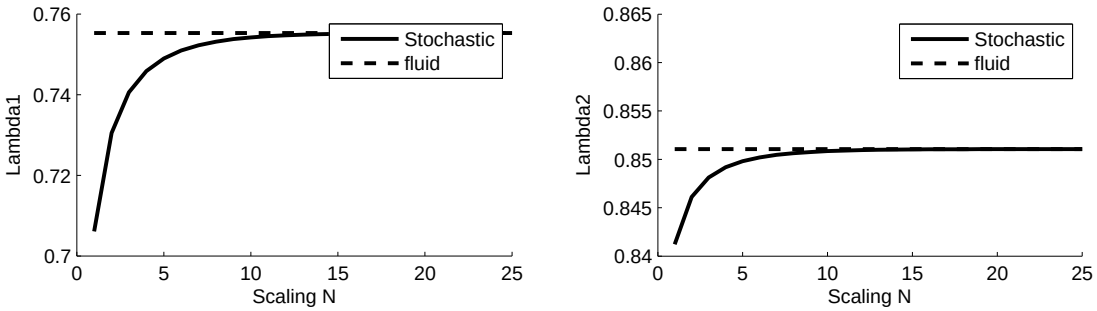


Figure 5.3: Both buffers empty, instance EE1

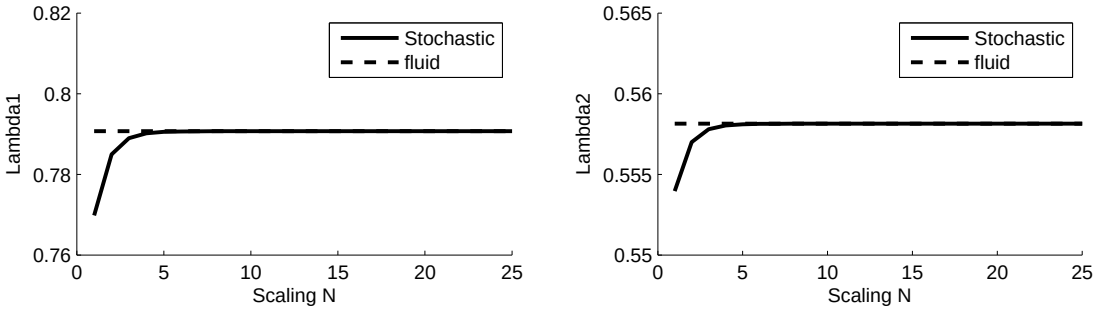


Figure 5.4: Both buffers empty, instance EE2

5.4.1 Both buffers empty

Instance EE1

$$\mu = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \alpha = \begin{pmatrix} 0.5 \\ 0.7 \end{pmatrix}, \quad P = \begin{pmatrix} 0 & 0.2 \\ 0.3 & 0 \end{pmatrix}, \quad Q = \begin{pmatrix} 0 & 0.1 \\ 0.2 & 0 \end{pmatrix}, \quad \lambda = \begin{pmatrix} 0.7553 \\ 0.8511 \end{pmatrix}.$$

Instance EE2

$$\mu = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \alpha = \begin{pmatrix} 0.4 \\ 0.4 \end{pmatrix}, \quad P = \begin{pmatrix} 0 & 0.7 \\ 0.2 & 0 \end{pmatrix}, \quad Q = \begin{pmatrix} 0 & 0.5 \\ 0.2 & 0 \end{pmatrix}, \quad \lambda = \begin{pmatrix} 0.5581 \\ 0.7907 \end{pmatrix}.$$

In Figures 5.3 and 5.4 it is observed that for both instances EE1 and EE2 λ_i^N converges nicely to the fluid solution λ_i as N increases. This means (5.3) holds when both buffers are empty. However, in this situation almost no rerouting will occur since both buffers are empty thus further investigation is required.

5.4.2 Both buffers full

Three cases are evaluated for this both buffers full situation because it is necessary to show that the formula for $\lambda_1^N(\mathrm{FF})$ is valid.

Instance FF1

$$\mu = \begin{pmatrix} 1 \\ 0.5 \end{pmatrix}, \quad \alpha = \begin{pmatrix} 0.9 \\ 0.7 \end{pmatrix}, \quad P = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \quad Q = \begin{pmatrix} 0 & 0 \\ 0.5 & 0 \end{pmatrix}, \quad \lambda = \begin{pmatrix} 1.5 \\ 1.7 \end{pmatrix}.$$

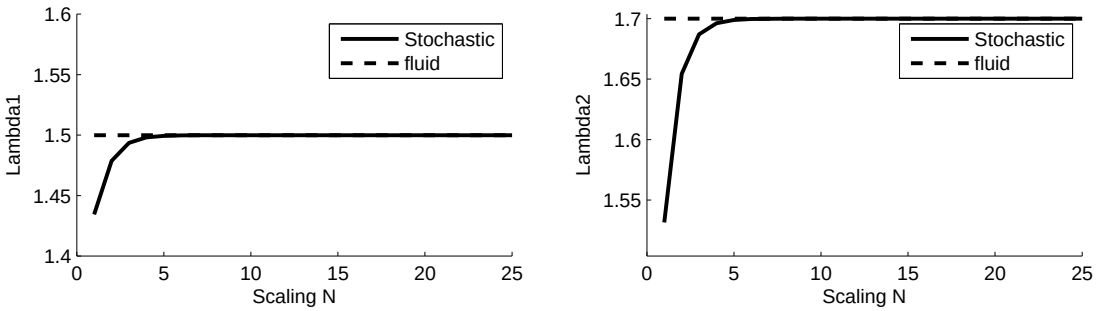


Figure 5.5: Both buffers full, instance FF1

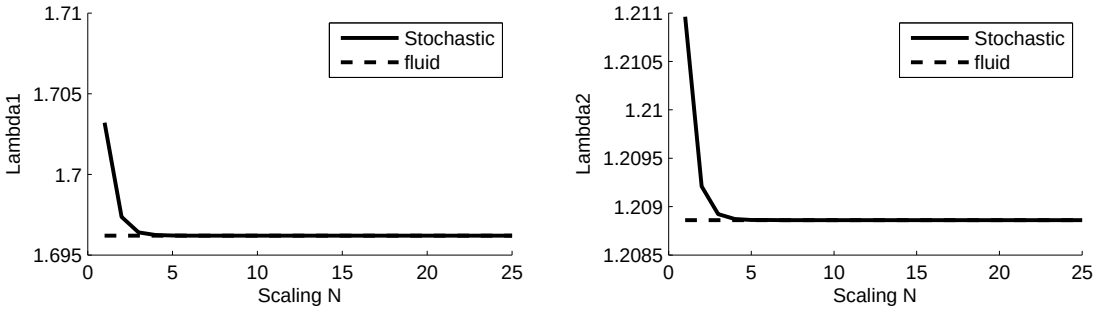


Figure 5.6: Both buffers full, instance FF2

Instance FF2

$$\mu = \begin{pmatrix} 1 \\ 0.5 \end{pmatrix}, \quad \alpha = \begin{pmatrix} 0.9 \\ 0.7 \end{pmatrix}, \quad P = \begin{pmatrix} 0 & 0.3 \\ 0.6 & 0 \end{pmatrix}, \quad Q = \begin{pmatrix} 0 & 0.3 \\ 0.7 & 0 \end{pmatrix}, \quad \lambda = \begin{pmatrix} 1.6962 \\ 1.2089 \end{pmatrix}.$$

Instances FF1 and FF2 (Figures 5.5 and 5.6) also look good and converge to their fluid solutions.

Instance FF3

As the parameters and the fluid solution for λ of instance FF3 (Figure 5.7) show, this is rather an extreme case of the both buffers full situation. At $N = 50$, the network spends a lot of time in state K_1, K_2: $\pi_{K_1,K_2} = 0.7570$. This is a large amount of time considering the system has 2601 states. A lot of 'ping-pong' will happen in this network. The convergence of λ_i^N for this network parameters to the fluid solution shows that the 'ping-pong' idea is true.

$$\mu = \begin{pmatrix} 0.2 \\ 0.5 \end{pmatrix}, \quad \alpha = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad P = \begin{pmatrix} 0 & 0.5 \\ 0.5 & 0 \end{pmatrix}, \quad Q = \begin{pmatrix} 0 & 0.7 \\ 0.9 & 0 \end{pmatrix}, \quad \lambda = \begin{pmatrix} 4.4973 \\ 4.10811 \end{pmatrix}.$$

5.4.3 One buffer full, one buffer empty

Instance EF1

$$\mu = \begin{pmatrix} 1 \\ 0.5 \end{pmatrix}, \quad \alpha = \begin{pmatrix} 0.5 \\ 0.7 \end{pmatrix}, \quad P = \begin{pmatrix} 0 & 0.3 \\ 0.1 & 0 \end{pmatrix}, \quad Q = \begin{pmatrix} 0 & 0.3 \\ 0.2 & 0 \end{pmatrix}, \quad \lambda = \begin{pmatrix} 0.6277 \\ 0.8883 \end{pmatrix}.$$

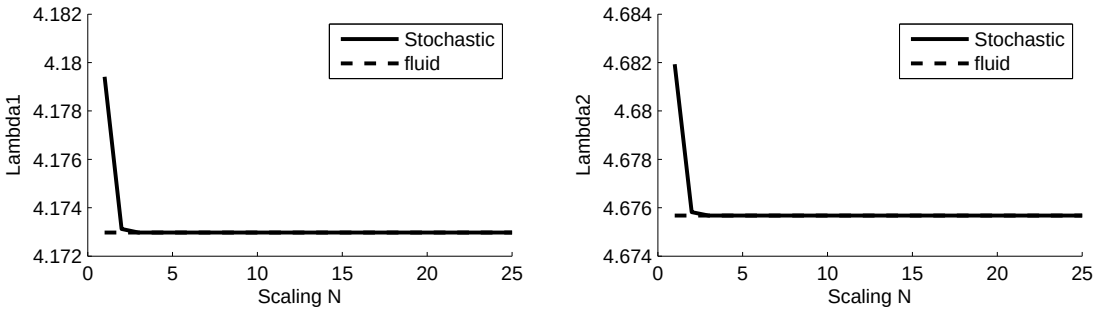


Figure 5.7: Both buffers full, instance FF3

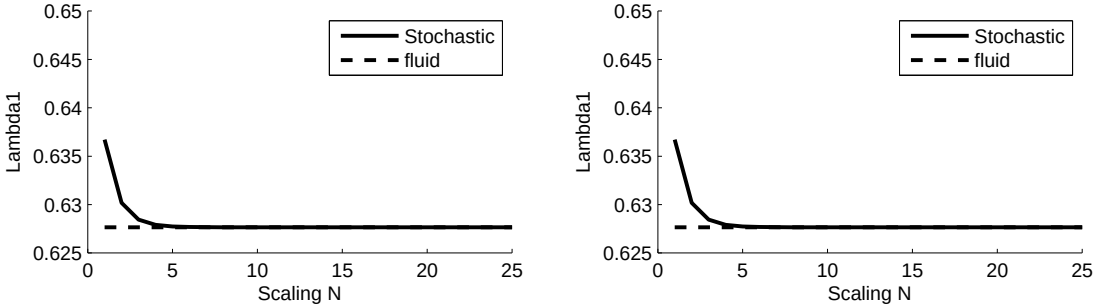


Figure 5.8: One buffer empty, one buffer full, instance EF1

Instance EF2

$$\mu = \begin{pmatrix} 1 \\ 0.5 \end{pmatrix}, \quad \alpha = \begin{pmatrix} 0.4 \\ 0.4 \end{pmatrix}, \quad P = \begin{pmatrix} 0 & 0.7 \\ 0.5 & 0 \end{pmatrix}, \quad Q = \begin{pmatrix} 0 & 0.5 \\ 0.5 & 0 \end{pmatrix}, \quad \lambda = \begin{pmatrix} 0.9231 \\ 1.0462 \end{pmatrix}.$$

In this case λ_i^N converges slower to the fluid solution than in the other cases. It is assumed this is caused by the fact that the fluid solution λ_i is close to μ_i and the time the system spends in each state is spread out more evenly over all the states.

5.5 Conclusion

All seven instances of the Jackson network with overflows show convergence of total inflow in a node in the discrete stochastic model to its fluid counterpart when scaling is increased. Because

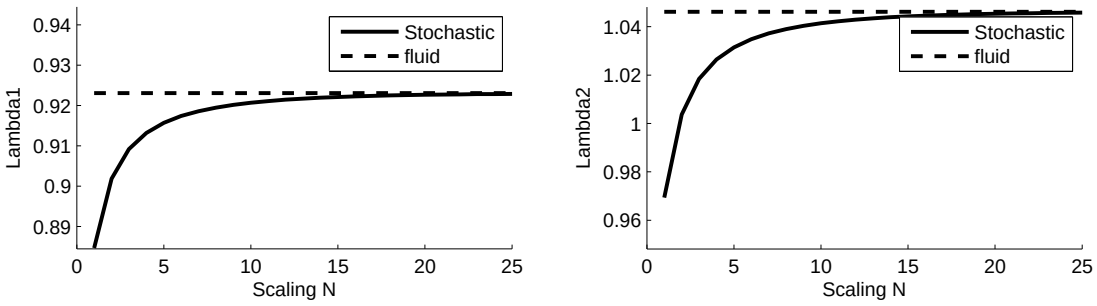


Figure 5.9: One buffer empty, one buffer full, instance EF2

the instances are very different and as there are no indications that there are possible instances for which (5.3) does not hold, it can be assumed that (5.3) does hold in general.

Chapter 6

Sojourn time distribution

In Chapter 3 a model to compute the sojourn time distribution is introduced. In this chapter its solutions for a network with one empty and one full buffer is compared to simulation results of the χ-model.

6.1 Approach

To describe the sojourn time distribution of a two node Jackson network with overflows it was first obtained by simulating the discrete stochastic network in χ for three different situations: both buffers full in the fluid solution, both buffers empty and one buffer full and one empty. Then an analytical approach, using the script described in section 3.3.2, was used to describe the sojourn time distribution for the fluid network in the situation where one node is full and one node is empty.

6.2 χ Simulations for the discrete stochastic case

At scaling 1, K_i^N is chosen to 10 for both nodes, since simulation time is limited and smaller buffers are more prone to overflowing in the stochastic case when they do not in the fluid case and vice versa. Of all simulations the sojourn times of the last $2 \cdot 10^7$ jobs before the simulation was stopped were used. Three situations are simulated, namely both buffers empty, both buffers full and one buffer empty and buffer full. Each situation is simulated for scaling 1 and 10.

6.2.1 Both buffers empty

The sojourn time distribution for a network instance with two empty buffers (Figure 6.1) shows a relatively constant slope for both scaling 1 and 10. Both buffers are mostly empty so jobs do not spend a lot of time waiting. In the fluid model of this instance of the Jackson network with overflows, jobs can have a sojourn time of 1 or 2 because $p_{21} = 0$ so jobs processed by node 2 never move to node 1.

$$\mu = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \alpha = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \quad P = \begin{pmatrix} 0 & 0.5 \\ 0 & 0 \end{pmatrix}, \quad Q = \begin{pmatrix} 0 & 0.5 \\ 0.5 & 0 \end{pmatrix}, \quad \lambda = \begin{pmatrix} 0.5 \\ 0.75 \end{pmatrix}.$$

6.2.2 Both buffers full

Figure 6.2 shows the sojourn time distribution of a network instance with two full buffers. For scaling 10 there is a lot more detail than scaling 1. For the fluid model, possible sojourn times are always a multiplication of 10 since a job spends always 10 time units in node 1 and 20 time

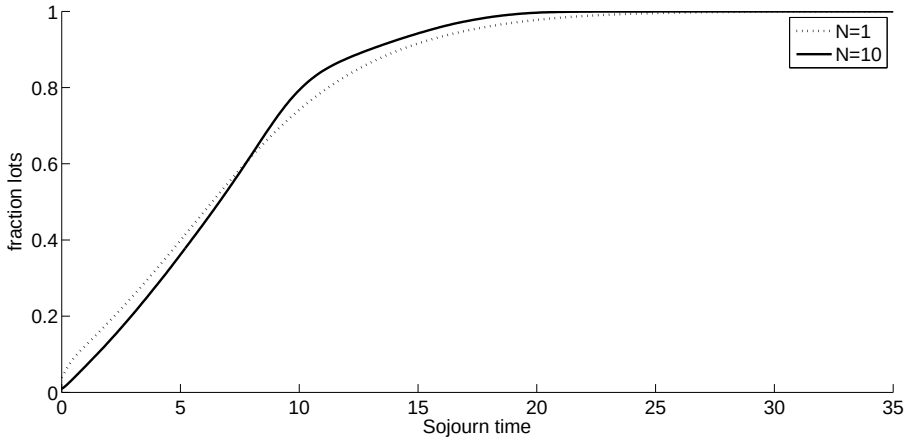


Figure 6.1: Both buffers empty

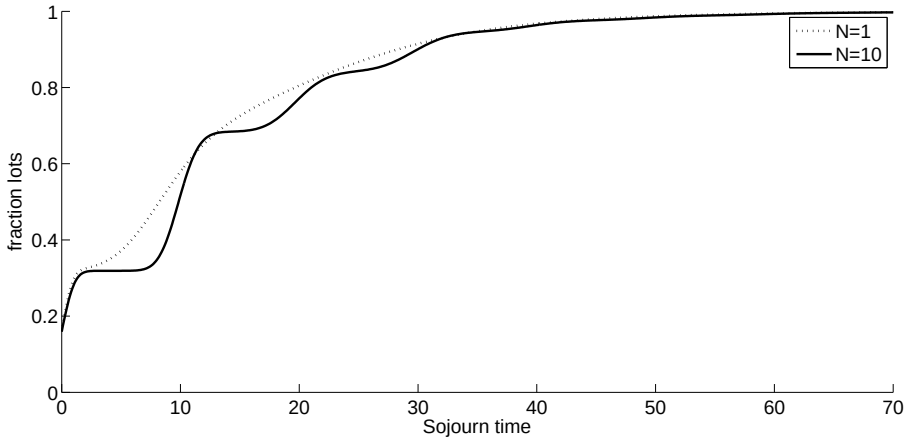


Figure 6.2: Both buffers full

units in node 2 when it has entered this node. Therefore the slope around $10, 20, \ldots$ is steep as it is caused by stochastic effects only.

$$\mu = \begin{pmatrix} 1 \\ 0.5 \end{pmatrix}, \quad \alpha = \begin{pmatrix} 0.95 \\ 0.7 \end{pmatrix}, \quad P = \begin{pmatrix} 0 & 0.3 \\ 0.6 & 0 \end{pmatrix}, \quad Q = \begin{pmatrix} 0 & 0.3 \\ 0.7 & 0 \end{pmatrix}, \quad \lambda = \begin{pmatrix} 1.6962 \\ 1.2089 \end{pmatrix}.$$

6.2.3 One buffer empty, one buffer full

When one buffer is empty and the other one is full, there are no slopes at $0, 10, 30, \ldots$ because the full node has waiting times of 20 and the slopes at $0, 20, 40, \ldots$ are not as steep as when both buffers are full (Figure 6.3) because these slopes are softened by the low processing time of the empty node.

$$\mu = \begin{pmatrix} 0.5 \\ 1 \end{pmatrix}, \quad \alpha = \begin{pmatrix} 0.4 \\ 0.4 \end{pmatrix}, \quad P = \begin{pmatrix} 0 & 0.5 \\ 0.7 & 0 \end{pmatrix}, \quad Q = \begin{pmatrix} 0 & 0.5 \\ 0.5 & 0 \end{pmatrix}, \quad \lambda = \begin{pmatrix} 1.0462 \\ 0.9231 \end{pmatrix}.$$

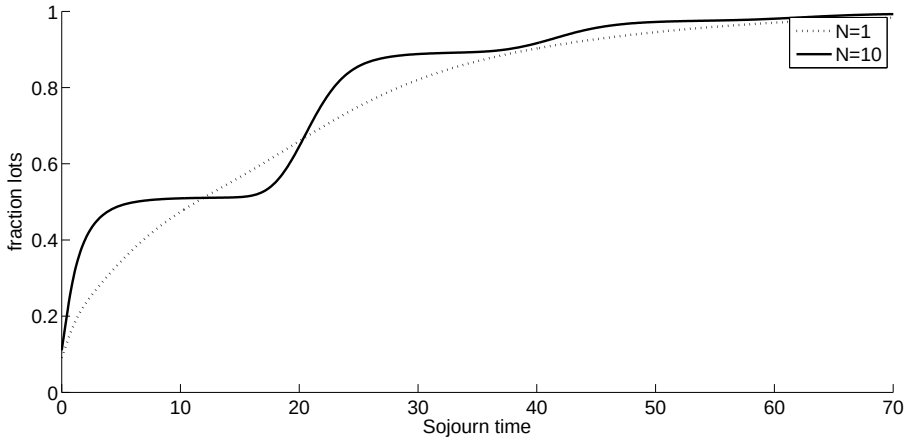


Figure 6.3: One buffer empty, one buffer full

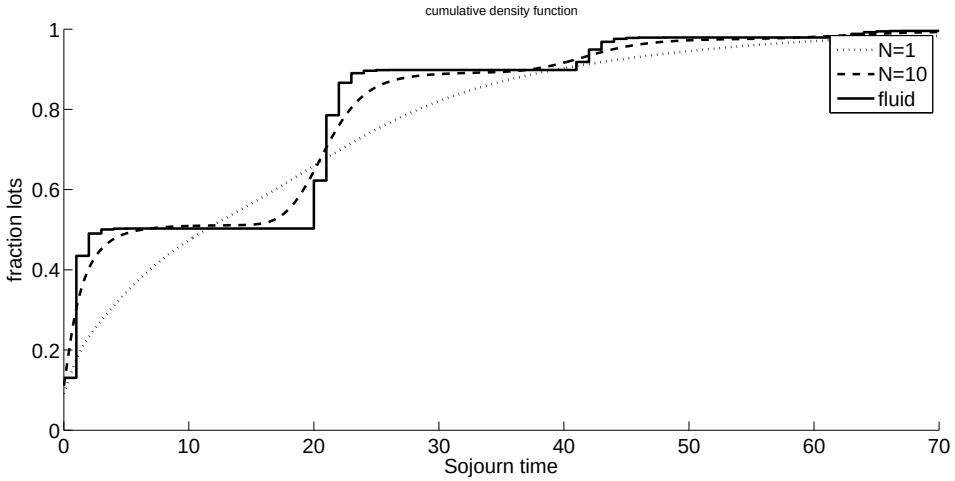


Figure 6.4: Sojourn time distribution for fluid and discrete stochastic

6.3 Comparing the discrete stochastic to the fluid model

Now the sojourn time distribution of the discrete stochastic instance of the Jackson network with overflow where one buffer is full and the other one empty is compared to its fluid counterpart.

In figure 6.4 it is seen that the sojourn time distribution of the 1 buffer full 1 empty situation for the fluid network compares very well to the discrete stochastic network. As scaling N for the discrete stochastic network increases the sojourn time distribution will probably converge to the sojourn time distribution of the fluid network since stochastic effects reduce, namely for an exponential distribution the variance decreases as the mean decreases (equation 6.1). Another indication for this conjecture to hold is given in Figures 6.1, 6.2, 6.3; networks with scaling 10 show more detailed sojourn time distribution compared to networks with scaling 1.

$$\sigma^2 = mean^2 = \frac{1}{(N * rate)^2}. \tag{6.1}$$

Chapter 7

Conclusion

The goal of this report was to answer the following question:
Do solutions for a discrete stochastic model of a Jackson Network with overflows converge, after scaling, to solutions for a continuous deterministic model of a Jackson Network with overflows?

And this question was divided in two parts:

1. Does the total inflow in a node in the discrete stochastic model converge after scaling to the inflow in a node in the fluid model?
2. Does the sojourn time distribution of the discrete stochastic model converge after scaling to the sojourn time distribution in the fluid model?

In chapter 5 it is show that (7.1) holds for several different instances of the Jackson network with rerouting. It can be safely assumed that (7.1) holds in general.

$$\lambda_i = \lim_{N \to \infty} \lambda_i^N \tag{7.1}$$

For the sojourn time distribution also convergence after scaling is observed. For future study it is recommended to also research instances of the network with two empty buffers and two full buffers. Also a greater scaling, for example $N = 500$ or $N = 1000$, is recommended since the sojourn time distribution does not converge as fast as the total inflows.

Bibliography

[1] R.A.A. Adams. *Calculus, a Complete Course*. Pearson, 2003.

[2] Stijn Fleuren. Dynamics of a finite buffer fluid overflow jackson network bachelor final project 420616. Master's thesis, department of Mechanical Engineering, Technical University of Eindhoven, Postbus 513 5600 MB Eindhoven, july 2010.

[3] V.G. Kulkarni. *Modeling, Analysis, Design, and Control of Stochastic Systems*. Springer, 1999.

Appendix A

χ Specification

In section 3.1 the χ-specification of a discrete stochastic model of a Jackson network with overflows is explained.

```
// Jackson Network with overflows

from standardlib import *
type lot = real

func det_dest(val u: real, Pi: 3*real) -> nat =
|[var j: nat = 0, uc: real = 0.0
:: (j < 3)*>
     ( uc:= uc + Pi.j
     ; (u <  uc -> ret j
       |u >= uc -> skip
       )
     ; j:= j + 1
     )
; ret 3            //3=exit
]|

proc G(chan a!: lot, val alpha: real, ID: nat) =
|[ var s: ->real, x: lot, t0: real
::( alpha <= 0.0 -> skip
| alpha >  0.0 -> s:= exponential(1/alpha)
                  ;*( x:= time; a!x
                    ; t0:= sample s; delay t0
                    )
)
]|

proc M(chan a?, b!: lot, val mu: real, ID: nat) =
|[ var s: ->real = exponential(1/mu), x: lot, t0: real
::*( a?x; t0:= sample s; delay t0; b!x )
]|

proc E(chan a?: lot) =
|[ var x: lot
:: *( a?x )
]|
```

```
proc B(chan a?,b?:3 # lot, c!:4 # lot, val P,Q:3*(3*real), K,X0: 3*nat) =
|[ var xs: 4*[lot], Kb: 4*nat, x:lot, u: -> real = uniform(0.0,1.0), dest: nat
 , outputs: 4*nat = <0, 0, 0, 0>  // "    "
::
(;, i: nat <- 0..3, xs.i:= [])        //empty buffers
; (;, i: nat <- 0..2, len(xs.i) < X0.i *> xs.i:= xs.i ++ [time] ) // fill buffers
; (;, i: nat <- 0..2, Kb.i:= K.i); Kb.3:= 100            //exit buffer to Kb
; *(
 // recieve from generator
 ( (|, i: nat <- 0..2, (a.i?x; dest:= i
                       ; (len(xs.dest) >= Kb.dest)
                            *> ( dest:= det_dest(sample u, Q.dest) )
                       ; xs.dest:= xs.dest ++ [x]
                       )
   )
 // recieve from machine
 | (|, i: nat <- 0..2, (b.i?x; outputs.i:= outputs.i+1
                       ; dest:= det_dest(sample u, P.i)
                       ; (len(xs.dest) >= Kb.dest)
                            *> ( dest:= det_dest(sample u, Q.dest) )
                       ; xs.dest:= xs.dest ++ [x]
                       )
   )
 | (|, i: nat <- 0..3, len(xs.i) > 0 -> c.i!hd(xs.i); xs.i:= tl(xs.i))
 )
 )
]|

proc S() =
|[ chan bm:4 # lot, mb,gb:3 # lot
 , var P: 3*(3*real) = < <0.0, 0.2, 0.4>   // Matrix P for routing
                       , <0.2, 0.0, 0.6>
                       , <0.4, 0.6, 0.0> >,
     Q: 3*(3*real) = < <0.0, 0.2, 0.4>   // Matrix Q for re-routing
                       , <0.2, 0.0, 0.6>
                       , <0.4, 0.6, 0.0> >,
     K: 3*nat      =   <10 , 10 ,  10>,  // Queue size K + 1 = Node capacity
     X0: 3*nat     =   <  5,   5,   5>,  // Starting value
     mu: 3*real    =   <1.0, 1.0, 1.0>,  // Process time mu
     alpha: 3*real =   <0.5, 0.5, 0.5>   // Inter-arrival time alpha
::
E(bm.3) || B(gb,mb,bm,P,Q,K,X0)
|| (||, i: nat <- 0..2, G(gb.i,alpha.i,i) || M(bm.i,mb.i,mu.i,i) )
]|

model Jackson() = |[ S() ]|
```

Appendix B

Markov chain Matlab function

Section 3.2 describes the Markov chain model of a discrete stochastic Jackson network with overflows. In this appendix it is given.

```
function [ Pi_mat ] = n2_jackson_overflow( alpha, mu, K, P, Q )
%N2_JACKSON_OVERFLOW function using Markov chain theory to solve the
% limiting probabilities for the discrete stochastic case of the Jackson
% network with rerouting
% Input: [alpha1 alpha2], [mu1 mu2], [k1 k2], [p12 p21], [q12 q21]
% Output: (K2+1,K1+1) state matrix

p12 = P(1); p21 = P(2);
q12 = Q(1); q21 = Q(2);
K1 = K(1); K2 = K(2);

G = zeros((K1+1)*(K2+1));
%{
StateSpace
S = {i,j, i=0, ..., K1, j=0, ..., K2}
Mapping Function
f_index(i,j) = 1+i+j*(K1+1)
%}

% Inside nodes
for i = 1:K1-1
    for j = 1:K2-1
        index = 1+i+j*(K1+1);
        G(index,index)      = -mu(1)-mu(2)-alpha(1)-alpha(2);
        G(index,index+1)    = alpha(1);
        G(index,index+K1+1) = alpha(2);
        G(index,index-1)    = (1-p12)*mu(1);
        G(index,index+K1)   = p12*mu(1);
        G(index,index-K1-1) = (1-p21)*mu(2);
        G(index,index-K1)   = p21*mu(2);
    end
end

% Left nodes (i empty)
i = 0;
for j = 1:K2-1
```

```
    index = 1+i+j*(K1+1);
    G(index,index)      = -mu(2)-alpha(1)-alpha(2);
    G(index,index+1)    = alpha(1);
    G(index,index+K1+1) = alpha(2);
    G(index,index-K1-1) = (1-p21)*mu(2);
    G(index,index-K1)   = p21*mu(2);
end

% Bottom nodes (j empty)
j = 0;
for i = 1:K1-1
    index = 1+i+j*(K1+1);
    G(index,index)      = -mu(1)-alpha(1)-alpha(2);
    G(index,index+K1+1) = alpha(2);
    G(index,index+1)    = alpha(1);
    G(index,index-1)    = (1-p12)*mu(1);
    G(index,index+K1)   = p12*mu(1);
end

% Right nodes (i full)
i = K1;
for j = 1:K2-1
    index = 1+i+j*(K1+1);
    G(index,index)      = -mu(1)-(1-p21*q12)*mu(2)-alpha(2)-q12*alpha(1);
    G(index,index-1)    = (1-p12)*mu(1);
    G(index,index+K1)   = p12*mu(1);
    G(index,index-K1-1) = (1-p21*q12)*mu(2);
    G(index,index+K1+1) = alpha(2)+q12*alpha(1);
end

% Top nodes (j full)
j = K2;
for i = 1:K1-1
    index = 1+i+j*(K1+1);
    G(index,index)      = -(1-p12*q21)*mu(1)-mu(2)-alpha(1)-q21*alpha(2);
    G(index,index-K1-1) = (1-p21)*mu(2);
    G(index,index-K1)   = p21*mu(2);
    G(index,index-1)    = (1-p12*q21)*mu(1);
    G(index,index+1)    = alpha(1)+q21*alpha(2);
end

% Corners
% State (0,0)
i = 0; j = 0;
index = 1+i+j*(K1+1);
G(index,index)      = -alpha(1)-alpha(2);
G(index,index+1)    = alpha(1);
G(index,index+K1+1) = alpha(2);

% State (0,K2)
i = 0; j = K2;
index = 1+i+j*(K1+1);
G(index,index)      = -alpha(1)-q21*alpha(2)-mu(2);
G(index,index+1)    = alpha(1)+q21*alpha(2);
```

```
G(index,index-K1-1) = (1-p21)*mu(2);
G(index,index-K1)   = p21*mu(2);

% State (K1,0)
i = K1; j = 0;
index = 1+i+j*(K1+1);
G(index,index)      = -alpha(2)-q12*alpha(1)-mu(1);
G(index,index-1)    = (1-p12)*mu(1);
G(index,index+K1)   = p12*mu(1);
G(index,index+K1+1) = alpha(2)+q12*alpha(1);

% State (K1,K2)
i = K1; j = K2;
index = 1+i+j*(K1+1);
G(index,index)      = -(1-p21*q12)*mu(2)-(1-p12*q21)*mu(1);
G(index,index-1)    = (1-p12*q21)*mu(1);
G(index,index-K1-1) = (1-p21*q12)*mu(2);


%Assemble equations
A=sparse([G ones((K1+1)*(K2+1),1)]); % add sum( PI(i) ) = 1
b=[zeros(1,(K1+1)*(K2+1))  1];
% Evalutate Pi*A = b
Pi_vect = b/A;

% order all states in a StateMatrix
Pi_mat = zeros(K2+1,K1+1);
for i= 0:K1
    for j= 0:K2
        index = 1+i+j*(K1+1);
        Pi_mat(j+1,i+1)= Pi_vect(index);
    end
end

end
```

Appendix C

Discrete stochastic flow calculation Matlab script

This appendix contains the Matlab script for the flow calculation from the limiting distribution given by the function in appendix B. The flow calculation is explained in section 5.2.1.

```
% lim_lambda_Ninf.m
% This file shows:
%      lim(N->inf)lambda_i_N = lambda_i_fluid
% functions clambda.m and n2_jackson_overflow.m are required.

clear all
close all

N_vect = 1:25;   %vector of scalings N

alpha = [.4 .4];   %starting value for alpha
mu = [1 .5];  %starting value for mu
P = [.7 .5];        %[p12 p21]
Q = [.5 .5];        %[q12 q21]

K = [2 2];

p12 = P(1);
p21 = P(2);
q12 = Q(1);
q21 = Q(2);

%solutions for lambda in the fluid case
lambda_fluid = clambda( alpha, mu, [0 p12; p21 0], [0 q12; q21 0] )

L1_N = [];
L2_N = [];

for N = N_vect
disp(sprintf('Scaling %d.', N))
Pi_mat = n2_jackson_overflow( N*alpha, N*mu, N*K, P, Q );
K1 = N*K(1);
K2 = N*K(2);
```

```
%EXPLICIT

% inside: no overflows and no empty buffers
% 0<X1<K1; 0<X2<K2
pi_S(1) = sum(sum( Pi_mat(2:K2, 2:K1) ));
lambda1_S(1) = alpha(1) + mu(2)*p21;
lambda2_S(1) = alpha(2) + mu(1)*p12;

%bottom: node2 empty
% 0<X1<K1; 0=X2
pi_S(2) = sum( Pi_mat(1, 2:K1) );
lambda1_S(2) = alpha(1);
lambda2_S(2) = alpha(2) + mu(1)*p12;

%left: node1 empty
% 0=X1; 0<X2<K2
pi_S(3) = sum( Pi_mat(2:K2, 1) );
lambda1_S(3) = alpha(1) + mu(2)*p21;
lambda2_S(3) = alpha(2);

%top: node2 full
% 0<X1<K1; X2=K2
pi_S(4) = sum( Pi_mat(K2+1, 2:K1) );
lambda1_S(4) = alpha(1) + mu(2)*p21 + alpha(2)*q21 + mu(1)*p12*q21;
lambda2_S(4) = alpha(2) + mu(1)*p12;

%right: node1 full
% X1=K1; 0<X2<K2
pi_S(5) = sum( Pi_mat(2:K2, K1+1) );
lambda1_S(5) = alpha(1) + mu(2)*p21;
lambda2_S(5) = alpha(2) + mu(1)*p12 + alpha(1)*q12 + mu(2)*p21*q12;

%top left: node1 empty, node2 full
% 0=X1; K2=X2
pi_S(6) = Pi_mat(K2+1, 1);
lambda1_S(6) = alpha(1) + mu(2)*p21 + alpha(2)*q21;
lambda2_S(6) = alpha(2);

%bottom right: node1,node2 full
% K1=X1; 0=X2
pi_S(7) = Pi_mat(K2+1, K1+1);
lambda1_S(7) = alpha(1)/(1-q12*q21) + alpha(2)*q21/(1-q12*q21) + ...
                                          mu(1)*p12*q21 + mu(2)*p21;
lambda2_S(7) = alpha(2)/(1-q21*q12) + alpha(1)*q12/(1-q21*q12) + ...
                                          mu(2)*p21*q12 + mu(1)*p12;

%bottom left: node1,node2 empty
% 0=X1; 0=X2
pi_S(8) = Pi_mat(1, 1);
lambda1_S(8) = alpha(1);
lambda2_S(8) = alpha(2);

%bottom right: node2 empty, node1 full
% K1=X1; 0=X2
```

```
pi_S(9) = Pi_mat(1, K1+1);
lambda1_S(9) = alpha(1);
lambda2_S(9) = alpha(2) + mu(1)*p12 + alpha(1)*q12;

L1_N = [L1_N sum(pi_S.*lambda1_S)];
L2_N = [L2_N sum(pi_S.*lambda2_S)];
end


figure(1)
hold on
plot(N_vect, L1_N, 'k','LineWidth',2)
plot([min(N_vect) max(N_vect)],[1 1]*lambda_fluid(1),'--k','LineWidth',2)
set(1,'OuterPosition',[10 10 400 300])
set(gca,'FontSize',10)
xlabel('Scaling N','FontSize',10)
ylabel('Lambda1','FontSize',10)
legend('Stochastic','fluid')

figure(2)
hold on
plot(L2_N,'k','LineWidth',2)
plot([min(N_vect) max(N_vect)],[1 1]*lambda_fluid(2),'--k','LineWidth',2)
set(2,'OuterPosition',[10 10 400 300])
set(gca,'FontSize',10)
xlabel('Scaling N','FontSize',10)
ylabel('Lambda2','FontSize',10)
legend('Stochastic','fluid')
```

Appendix D

Fluid flow calculation

In this appendix the Matlab conversion from Stijn Fleurens Mathimatica function is given [2]. It is briefly described in section 3.3.1.

```
function [ lambda ] = clambda( a, mu, P, Q )
%CLAMBDA, converted from Stijn Fleuren's Mathematica script.
% Calculates lambda for the fluid case of the
%  Jackson network with overflows
%  input: [a1 a2 ... a_n], [mu1 mu2 ... mu_n] P(n,n), Q(n,n)
NN = length(P);           %(*network size*)
%(* calculation 2 in flowchart*)
   B = zeros(1,NN);
   endloop1 = false;
   while endloop1 == false
       %(*calculation 3 in flowchart*)
        G = zeros(1,NN);
        endloop2 = false;
        while endloop2 == false
            %(*calculation 4 in flowchart*)
    SG = diag(G);
    SB = diag(B);
    lambda = ...
    (inv(eye(NN) - (SG*P)' - (SB*Q)'))*...
         ((((eye(NN) - SG)*P)'*mu') - ((SB*Q)'*mu') + a');
            %(*check 5 in flowchart*)
    y = 0;
    for i = 1:NN,
     if (G(i) == 1 && lambda(i) <= mu(i)) || (G(i) == 0 && lambda(i) > mu(i))
        y=y+1;
     else
         break
     end
     if y == NN
         endloop2 = true;
     end
    end
            %(*calculation 6 in flowchart*)
    for i = 1:NN
     if lambda(i) > mu(i)
         G(i) = 0;
```

```
     else G(i) = 1;
     end
    end
    end
        %(*check 7 in flowchart*)
      if (G + B) == ones(1,NN)
          endloop1 = true;
      end
  %(*calculation 8 in flowchart*)
      for i = 1:NN,
   if G(i) == 0
       B(i) = 1;
   else
       B(i) = 0;
   end
      end
   end
%(*calculation 9 in flowchart*)
EmptyB = [];
FullB = [];
UnknownB = [];
for i = 1:NN
 if G(i) == 1 && lambda(i) == mu(i)
     UnknownB = [UnknownB i];
 else
  if G(i) == 1 && lambda(i) < mu(i)
      EmptyB = [EmptyB i];
  else
   FullB = [FullB i];
  end
 end
end
lambda=lambda';
```

Appendix E

Fluid sojourn time distribution script

The script below together with the function in section E.1 computes the cumulative density function of the sojourn time distribution. Both are described in section 3.3.2.

```
% Script computes the CDF of sojourn times for
%   the fluid deterministic Jackson network with overflows
%   Node 1 must be full and Node 2 must be empty

% function files rfunc.m and clambda.m are needed.

close all
max_t = 30;            %maximum time (stopping criterium)

% Network Parameters
alpha =  [1 .5];
mu =     [.5 2];
K =      [10 10];
P =      [0.5 0.5];    % [p12 p21]
Q =      [0.0 0];    % [q12 q21]

lambda = clambda(alpha,mu,[0 P(1); P(2) 0],[0 Q(1); Q(2) 0])

% lambda1>mu1; lambda2<mu2 ???
if lambda(1)<=mu(1) || lambda(2)>=mu(2)
    error('lambda1<=mu1 or lamdba(2)>=mu(2). Adapt your network parameters. Buffer 1 should be fu
end

% V = [Time vector, Fluid leaving vector]
% Node 1
% Fluid leaving at t=0
V = [0 alpha(1)*(1-mu(1)/lambda(1))*(1-Q(1))];

% fluid entering node1:
%   alpha(1)*mu(1)/lambda(1)
V = [V; rfunc(1, alpha(1)*mu(1)/lambda(1), 0, alpha, ...
                                mu, lambda ,K, P, Q, max_t)];

% Node 2
```

```
% fluid entering node 2:
%   alpha(2)
%   alpha(1)*(1-mu(1)/lambda(1))*Q(1)       (rerouted from node1)
V = [V; rfunc(2, alpha(2)+alpha(1)*(1-mu(1)/lambda(1))*Q(1), 0, ...
                                  alpha, mu, lambda ,K, P, Q, max_t)];

%normalize V
V(:,2) = V(:,2)./sum(alpha);
check1 = sum(V(:,2))           %check, should be 1 (Fcdf(inf)=1)
V = sortrows(V);               %sort V on time (ascending)
%make cumulative
for i = 2:length(V)
    V(i,2) = V(i-1,2)+V(i,2);
end

stairs(V(:,1),V(:,2))
title('cumulative density function')
xlabel('Sojourn time','FontSize',16)
ylabel('fraction lots','FontSize',16)
set(gca,'FontSize',16)
set(1,'OuterPosition',[50 50 800 600])
```

E.1 Recursive function

```
function V = rfunc( node, frac_in, t, alpha, mu, lambda, K, P, Q, max_t )
%RFUNC recursive function that evaluates all routes a fluid molecule can
% possibly do untill reaching time max_t
% Input :
%   node       previous node
%   frac_in    fraction of fluid just processed in previous node
%   t          current time
%   [alpha1 alpha2], [mu1 mu2], [lambda1 lambda2],
%   [k1 k2], [p12 p21], [q12 q21]
%   max_t      maximum time (stopping criterium)
% Output:
%   V = [Time vector, Fluid leaving vector]

if node == 1
    %leaving node 1:
    %   exit 1-P12
    %   enter node2 P12

    e = frac_in * (1-P(1));
    t = t + K(1)/mu(1);
    if (t < max_t)
        V = [t e; rfunc(2, frac_in*P(1), t, alpha, mu, ...
                                    lambda, K, P, Q, max_t)];
    else
        V = [t e];
    end
elseif node == 2
    %leaving node 2:
    %   exit            1-P21
```

```
    %   exit            P21*(1-mu(1)/lambda(1))*(1-Q12)
    %   enter node1     P21*(mu(1)/lambda(1))
    %   enter node2     P21*(1-mu(1)/lambda(1))*Q12

    e = frac_in * (1-P(2) + P(2)*(1-mu(1)/lambda(1))*(1-Q(1)));
    t = t + 1/mu(2);
    if (t < max_t)
        V = [t e; rfunc(1, frac_in*P(2)*(mu(1)/lambda(1)), ...
                           t, alpha, mu, lambda, K, P, Q, max_t);...
                  rfunc(2, frac_in*P(2)*(1-mu(1)/lambda(1))*Q(1), ...
                           t, alpha, mu, lambda, K, P, Q, max_t)];
    else
        V = [t e];
    end
end

end
```