

UNIVERSITEIT UTRECHT

---

# Analyzing a queueing network

---

*Author:*

William SCHELLING

*Supervisor:*

dr. Sandjai BHULAI

dr. Karma DAJANI

*A thesis submitted in fulfillment of the requirements  
for the degree of Master of Science*

*in the*

Department of Mathematics

Universiteit Utrecht

August 2014



# *Abstract*

The purpose of the thesis is to help ING Bank to get a better understanding of their ICT landscape. Since the ICT landscape of ING Bank is very large, we will focus on a specific part of the ICT landscape. ING Bank wants to identify the bottlenecks more quickly. Therefore we develop two different mathematical tools which can help in identifying the bottlenecks of such a complex queueing system. The first tool is a simulation tool using the program Rockwell Arena. The second tool is an analytical approach, based on Markov chains. Both tools have some advantages and disadvantages which we also discuss in the thesis.



## *Acknowledgements*

Instead of making a pure theoretical thesis, I decided to work on a practical problem. I want to thank ING Bank, in particular Joost Bosman, to provide a practical problem. I would also like to thank Sandjai Bhulai for his supervision during this thesis. He was a great help with his insights. I also like to thank Karma Dajani for being the supervisor of University Utrecht. I also want to thank Cristian Spitoni for being the second reader. At last but not least I want to thank my parents for supporting me during the whole study.



# Contents

|   |            |
|---|------------|
| <b>Abstract</b>   | <b>iii</b> |
| <b>Acknowledgements</b>   | <b>v</b>   |
| <b>Contents</b>   | <b>vi</b>  |
| <b>List of Figures</b>  | <b>ix</b>  |
| <b>List of Tables</b>   | <b>xi</b>  |
| <b>1 Introduction</b>   | <b>1</b>   |
| 1.1 Introduction . . . . .                                      | 1          |
| <b>2 Local Order Manager</b>                                    | <b>3</b>   |
| 2.1 What is the Local Order Manager? . . . . .                  | 3          |
| 2.2 What is the problem with the Local Order Manager? . . . . . | 5          |
| 2.2.1 Service: CreateOrder . . . . .                            | 6          |
| <b>3 Mathematical background</b>                                | <b>9</b>   |
| 3.1 Queueing theory . . . . .                                   | 9          |
| 3.1.1 Kendall's notation . . . . .                              | 9          |
| 3.1.2 Exponential distribution . . . . .                        | 11         |
| 3.1.3 Poisson process . . . . .                                 | 13         |
| 3.1.4 PASTA . . . . .   | 15         |
| 3.1.5 (Limited) Processor Sharing . . . . .                     | 16         |
| 3.1.6 Sessions . . . . .  | 16         |
| 3.1.7 Time to live . . . . .                                    | 17         |
| 3.1.8 Important measures . . . . .                              | 17         |
| 3.1.9 Little's law . . . . .                                    | 18         |
| <b>4 Simulation</b>   | <b>19</b>  |
| 4.1 Rockwell Arena . . . . .                                    | 20         |
| 4.1.1 A M/M/1 model in Arena . . . . .                          | 21         |
| 4.1.2 Gather statistics . . . . .                               | 22         |
| 4.1.3 Maximum queue length in Arena . . . . .                   | 23         |
| 4.1.4 (Limited) Processor sharing in Arena . . . . .            | 23         |
| 4.1.5 Sessions in Arena . . . . .                               | 25         |

|          |  |           |
|----------|--|-----------|
| 4.1.6    | Time to live in Arena                    | 25        |
| 4.2      | CreateOrder in Arena                     | 27        |
| 4.2.1    | Arrival pattern and queue                | 27        |
| 4.2.2    | Splitting                                | 28        |
| 4.2.3    | IOH 1                                    | 28        |
| 4.2.4    | PIM                                      | 29        |
| 4.2.5    | IOH 2                                    | 30        |
| 4.2.6    | AOC                                      | 30        |
| 4.2.7    | IOH 3                                    | 31        |
| 4.2.8    | OR                                       | 31        |
| 4.2.9    | IOH 4                                    | 31        |
| 4.2.10   | Variables                                | 32        |
| 4.3      | Bottleneck analysis                      | 33        |
| <b>5</b> | <b>Markov chains</b>                     | <b>39</b> |
| 5.1      | Markov chain                             | 39        |
| 5.1.1    | Properties                               | 41        |
| 5.1.2    | The limiting distribution                | 42        |
| 5.1.3    | Time reversible                          | 46        |
| 5.2      | Continuous-time Markov chain             | 48        |
| 5.2.1    | The limiting distribution                | 49        |
| 5.2.2    | Embedded Markov Chain                    | 52        |
| 5.2.3    | Time reversal                            | 53        |
| 5.3      | Queueing theory                          | 54        |
| 5.3.1    | M/M/1                                    | 54        |
| 5.3.1.1  | Processor sharing                        | 56        |
| 5.3.1.2  | Other simple queueing networks           | 57        |
| 5.3.2    | More complex queueing networks           | 58        |
| 5.3.2.1  | n M/M/1 queues in parallel               | 58        |
| 5.3.2.2  | n M/M/1 queues in series                 | 59        |
| 5.3.3    | Even more complex networks               | 61        |
| 5.4      | Solving the CreateOrder queueing network | 62        |
| 5.4.1    | Bottleneck analysis                      | 65        |
| <b>6</b> | <b>Comparison</b>                        | <b>69</b> |
| 6.1      | Comparing the results                    | 71        |
| <b>7</b> | <b>Final remarks</b>                     | <b>73</b> |
| 7.1      | Conclusion                               | 73        |
| 7.2      | Future research                          | 74        |
| <b>A</b> | <b>Mathematica code</b>                  | <b>77</b> |
|          | <b>Bibliography</b>                      | <b>81</b> |



# List of Figures

|      |  |    |
|------|--|----|
| 2.1  | LOM functional modules . . . . .   | 4  |
| 2.2  | Deployment view . . . . .  | 5  |
| 2.3  | CreateOrder flow . . . . .   | 7  |
| 3.1  | The M/M/1 queue in graphical notation. . . . .                           | 11 |
| 3.2  | Graphical notation of sessions. . . . .                                  | 17 |
| 4.1  | M/M/1 in Arena . . . . .   | 22 |
| 4.2  | M/M/1/n in Arena . . . . .   | 23 |
| 4.3  | Processor sharing in Arena . . . . .                                     | 24 |
| 4.4  | Sessions in Arena. . . . .   | 25 |
| 4.5  | Time to live applies on a queue . . . . .                                | 26 |
| 4.6  | Complete model . . . . .   | 27 |
| 4.7  | The arrival pattern and the queue . . . . .                              | 28 |
| 4.8  | Splitting . . . . .  | 29 |
| 4.9  | IOH1 . . . . .   | 30 |
| 4.10 | PIM . . . . .  | 30 |
| 4.11 | AOC . . . . .  | 31 |
| 4.12 | OR . . . . .   | 31 |
| 4.13 | Sessions ending . . . . .  | 32 |
| 4.14 | Blocking under normal scenario . . . . .                                 | 36 |
| 4.15 | Blocking at incoming queue compared to blocking at PIM . . . . .         | 36 |
| 4.16 | Blocking under the standard model and 5 different scenarios . . . . .    | 37 |
| 5.1  | n parallel M/M/1 queues . . . . .  | 59 |
| 5.2  | n M/M/1 queues in series . . . . .                                       | 60 |
| 5.3  | A queueing system containing three M/M/1 queues. . . . .                 | 61 |
| 5.4  | The queueing network corresponding with the CreateOrder service. . . . . | 63 |
| 5.5  | Blocking probability and throughput in the current situation . . . . .   | 66 |
| 5.6  | Blocking probability and throughput under some scenarios . . . . .       | 66 |
| 5.7  | Blocking probability and throughput under some other scenarios . . . . . | 67 |
| 5.8  | Blocking probability after adding an extra PIM-server. . . . .           | 67 |
| 5.9  | Expected number of customers. . . . .                                    | 67 |
| 5.10 | Expected number of customers after doubling the sessions. . . . .        | 68 |



# List of Tables

|     |   |    |
|-----|---|----|
| 4.1 | Arena . . . . .   | 21 |
| 4.2 | Our approach for a M/M/1-PS queue in Rockwell Arena under different step sizes (from 10 to 0.01) compared to the theoretic result. $\mu = 1$ , different values for $\lambda$ . . . . . | 24 |
| 4.3 | Variables that let the process run smoothly . . . . .   | 33 |
| 4.4 | Input variables . . . . .   | 34 |
| 4.5 | Record variables: Tally statistics. . . . .   | 34 |
| 4.6 | Record variables: Counters . . . . .  | 35 |



# Chapter 1

## Introduction

### 1.1 Introduction

The ING Bank has to keep track of the performance of the ICT landscape of the Bank. The ING Bank has to provide good services for its customer. It would be a disaster if *Mijn ING Internet banking* is out of the air for many times. That would be crucial for the reputation of the Bank and thus also for the business. Therefore ING Bank wants a better understanding of what happens in the ICT landscape of the bank. They want to know if the current settings are the best. They will also be prepared for the future. Therefore they want to answer questions such as 'Where is the bottleneck in our system?', 'What happens with the system under some scenarios?', 'How can we anticipate in time the bottlenecks?', ...

Motivated by this, they want a mathematical model which can answer such kind of questions. Since the ICT landscape is very complex and large, they have decided to first investigate the Local Order Manager (LOM), a key point in the network. The Local Order Manager handles millions and millions of requests every day. We will introduce the LOM further in Chapter 2. Even the LOM itself is very complex. Therefore we will mainly focus on 1 specific service of the LOM: *The CreateOrder service*.

We will try to model the CreateOrder and analyze this model to determine the bottlenecks. We will do this with two approaches. The first approach is a simulation where we use the program Rockwell Arena. The second approach is by using results of Markov chains and Continuous-time Markov chains.

In Chapter 2 we will have a closer look at the LOM and the Create Order service and we will also discuss the main problems they encounter with the LOM. We will also see that the CreateOrder service can be seen as queueing model. Therefore we use Chapter 3 to explain some vocabulary used in the mathematical field of queueing networks. We

will for example explain the meaning of Kendall's notation, which we will use in other chapters very often.

Chapter 4 is devoted to the simulation approach. We will discuss the program Rockwell Arena and how we can use this program to simulate and analyze a queueing network. At the end of the Chapter we come up with a model for the CreateOrder service and we will analyze this model.

Chapter 5 is devoted to the Markov approach. We will provide a mathematical foundation based on (Continuous-time) Markov chains which can be applied to queueing networks. We develop some tools to analyze a queueing system analytically. At the end of this Chapter we provide a model of the CreateOrder service and we will analyze this model with the developed tools.

Chapter 6 is devoted to compare both approaches. We will not only compare the results of the two approaches with each other, but we will also compare the approaches in general. We will discuss the advantages and disadvantages of both approaches.

Chapter 7 is the last Chapter where the conclusions are made. We will look back at the results of this thesis and state some possible future research directions.

## Chapter 2

# Local Order Manager

### 2.1 What is the Local Order Manager?

In this chapter we will introduce the Local Order Manager (LOM). What is it? How does it work? And what are the problems? The LOM is a part of the ICT landscape of the ING Bank. The LOM is responsible for a part of the domestic financial orders through the system. I will give a simple example: Assume you are a customer of the ING Bank and you have to pay a bill. So you log in on E-banking and pay the bill. In fact you send a request to the LOM to handle this transaction. The LOM handles the process through the whole ICT landscape. It determines the whole path through the system and also follows the job through the system. Roughly speaking the LOM is responsible for following different tasks. It validates and enriches financial orders. An example of this is adding the Leading IBAN to the job. It follows the orders until they are executed. So if you now plan a transaction, which must be executed over 5 days, the LOM is responsible to store the request for 5 days and it is responsible for the execution at day 5. The LOM releases orders for execution and routes them to the payment engine. So if the job is ready for execution, the LOM releases the jobs and gives the right payment engine the task to fulfill the job. The LOM also tracks the status of an order during the entire life-cycle of the job. So the LOM knows where each job is.

It will not be a surprise that the LOM consists of different modules, with their own functionality. Figure 2.1 shows us an overview of the different modules the LOM can do. Everything within the yellow box is part of the LOM. As you see, there are also some boxes outside the yellow box. These are other servers. They could send messages to LOM or receive messages from LOM. Or in some cases both. One of the most important box outside the LOM is 'MING-P, MING-Z, Mobiel, Saldolijn, Channel-App, STOE, Ringo, Lisa, SavSolSel, Hypos, Fundation, Profile, etc.' standing above the LOM. These are

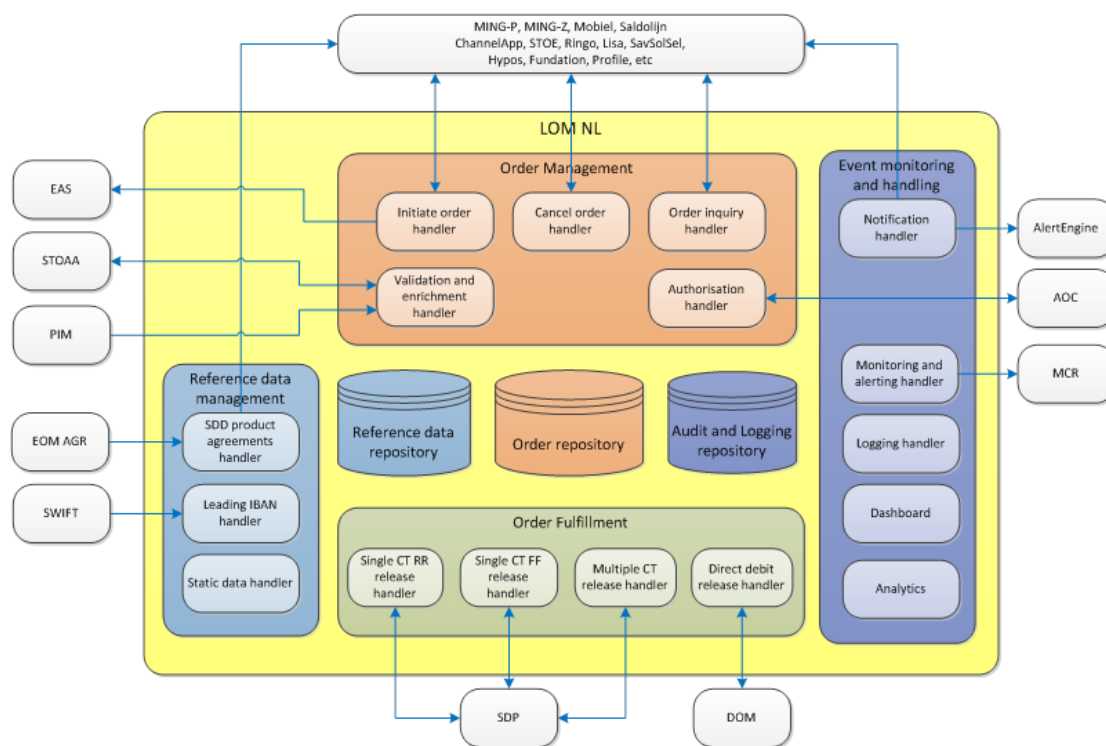


FIGURE 2.1: LOM functional modules

the incoming channels.

In order to handle all tasks, the LOM is divided in 16 nodes. Each node can handle some tasks. Which tasks a node can handle is determined by the services. A service corresponds to a task, for example: 'Create a new order'. There are around 40 services. These services are divided in groups. The groups are assigned to some nodes. Each node can handle the services of up to 4 groups. Each group is assigned to multiple nodes. Figure 2.2 shows another representation of the LOM, based on the 16 nodes.

Each node has 16 threads. Roughly speaking this means that there are at most 16 jobs in service at the same time in a specific node. Each busy thread stands for 1 job in the system. Each service also has his own number of listeners. If a service has 5 listeners in a specific node, then that means that the specific service can be executed for 5 listeners at the same time. So if a new job arrives, it can be served if

1. There is a node with a thread that is not busy.
2. The job can be executed on that node. So the specific service needed for the job is assigned to that node.
3. There is a free listener for that specific service.



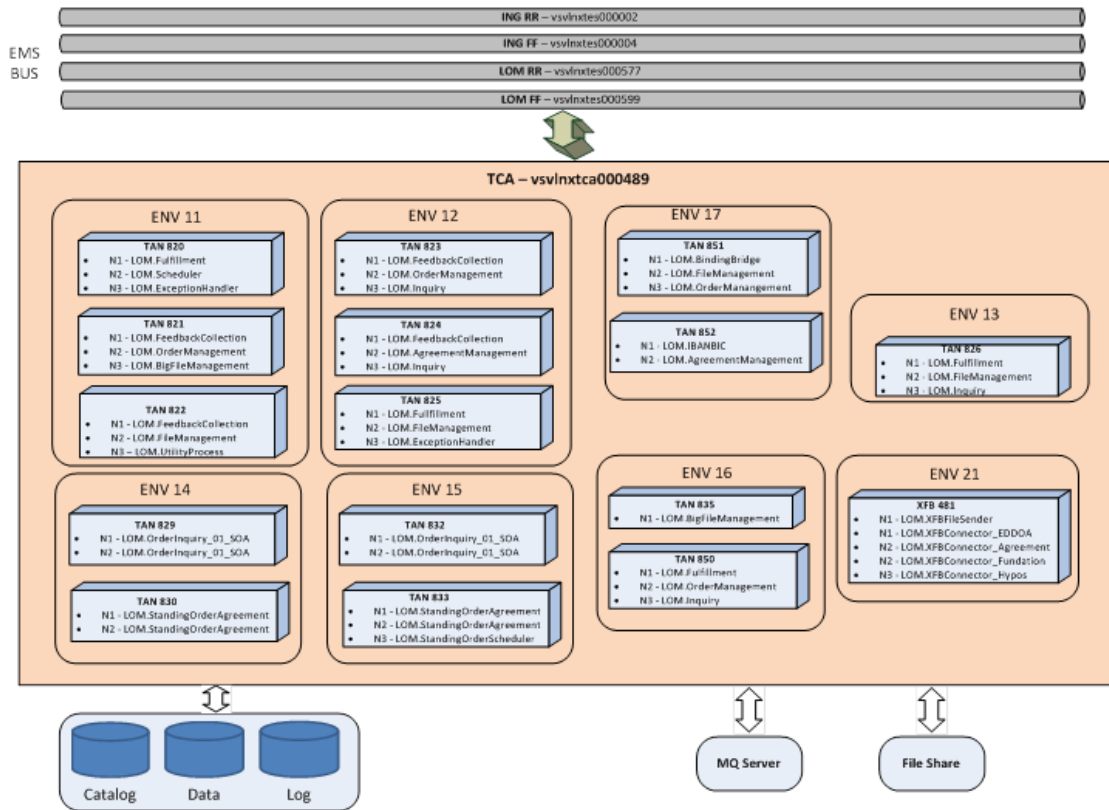


FIGURE 2.2: Deployment view

If these requirements are not met, the job has to wait in a queue before it can enter the LOM. From a mathematical point of view we can see the LOM as a queuing system, with some different servers, some different job types and some constraints.

## 2.2 What is the problem with the Local Order Manager?

In the previous section we introduced the LOM. We have seen that there are some parameters. For example the number of listeners and the number of threads. The ING Bank does not know the best values for the parameters. They change the parameters by trial and error and hope that the performance increases. That is not an ideal situation. Some question are:

1. What is the right number of listeners for a service?
2. What is the right number of threads for a node?
3. Which services have to be assigned to which node?
4. What is the current bottleneck in the LOM?

Unfortunately, it is not possible to model the LOM completely. Because it is too big. For that reason we will focus on modeling and analyzing a specific service. We will do that from two different points of view:

1. Using a simulation tool.
2. Using a theoretical approach.

We will discuss each step in both approaches extensively, so the ING Bank can use this thesis as a guide for modeling the remaining services. Therefore we will not only discuss the steps of creating the model, but also the steps of analyzing the model.

### 2.2.1 Service: CreateOrder

In this thesis our focus is on the CreateOrder service. The CreateOrder Service is one of forty services inside the LOM and one of the most used services. Therefore it is important to model this service first. Figure 2.3 shows the flow through the LOM and some other machines which are called in the process. We will try to model each server used by the CreateOrder service. The four servers are the Initiate Order Handler (part of the LOM), AOC, PIM and Order repository (part of the LOM). The process starts at the IOH. IOH sends a request to PIM, so that will be the next step. After some time, IOH gets the reply and sends immediately a request to AOC. AOC sends the request back to the IOH. IOH gets the reply and sends immediately a request to OR. After OR sends a reply to IOH, IOH sends a reply to the customer and the process is done.

The CreateOrder service has 15 listeners per node and is deployed on 4 different nodes. Since CreateOrder is a request-reply service, the Listener is busy from beginning to the end. This means that there can only be  $4 \times 15$  jobs in service at some moment. If there are more the 60 jobs in the system, they have to wait in the queue of the LOM.

Each server has its own characteristics. Each server can handle a certain number of jobs at the same time. Each server has its own queue. Some servers are deployed on one node, others are deployed on multiple nodes. These characteristics determine the performance of the system. For example: If the process is deployed on more nodes, the required time will decrease.

In both approaches we will try to model the different services and the characteristics as well as possible and we try to say something about the behavior of the system under different kind of loads. We will try to say something about some scenarios in which we adapt the model a bit.

## Sequence Diagram

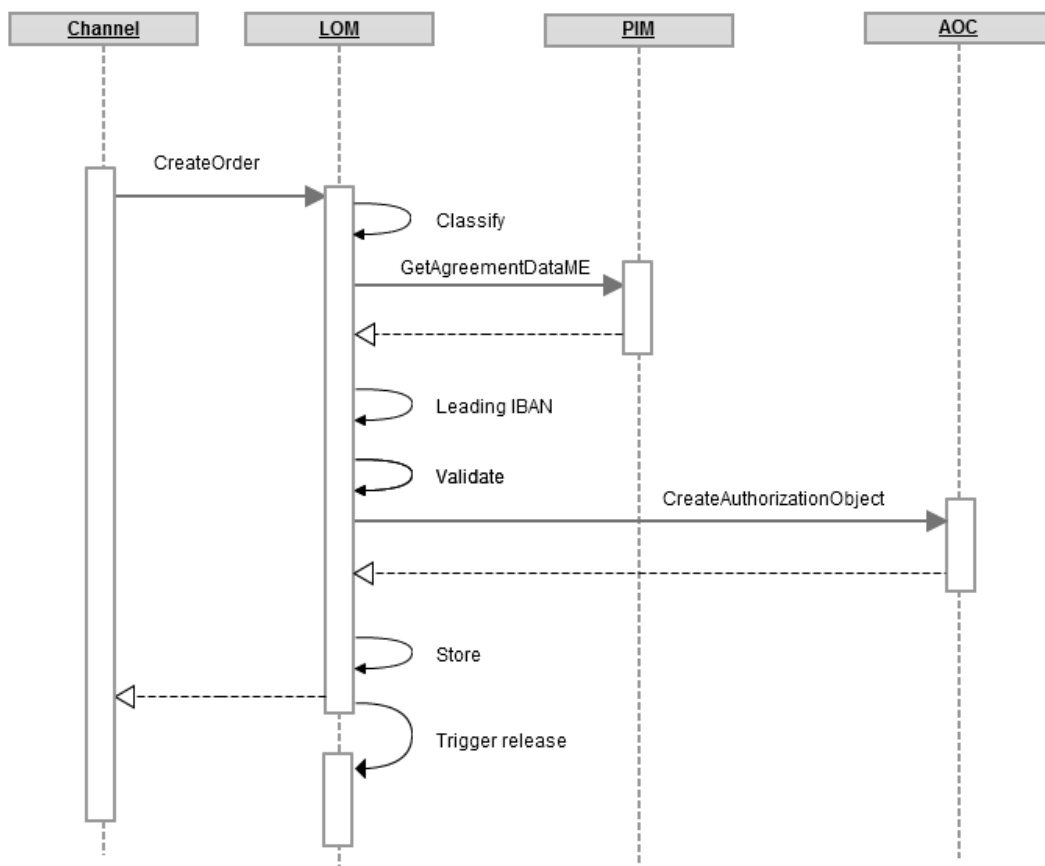


FIGURE 2.3: CreateOrder flow



## Chapter 3

# Mathematical background

In the previous chapter we have seen that the LOM can be seen as a complex queueing system. In this chapter we will provide a mathematical background for the concept of queueing theory. We will mainly focus on the vocabulary used in queueing theory. Since queueing theory is a broad discipline, we will focus on the things that are really necessary.

Queueing theory is the mathematical study of queueing systems. Queueing systems can be seen everywhere, in manufacturing, in traffic, in computer networks. Mathematics will try to answer questions about the behavior of queueing systems. How is the performance of the system? Can we say something about the number of jobs in the system? Can we say something about the time a job spends in the system? Can we predict something about the bottleneck of the system?

### 3.1 Queueing theory

#### 3.1.1 Kendall's notation

One of the easiest systems to analyze is M/M/1 queue. To understand what the M/M/1 queue means we will have a look at Kendall's notation. This is a universal way to describe the configuration of most queueing systems. Kendall's notation [1] is based on 6 slots: A/S/c/K/N/D. Each slot gives us information about the queueing system. We will briefly discuss the 6 slots.

1. A - This letter specifies the arrival pattern. The most common letters in this slot are M, D and G. The letter G stands for general distribution. You want to try to answer the questions about the behavior of the system without any assumption on

the arrival pattern. The D stand for deterministic. In this case the time between two arrivals is a fixed time unit. there will be for example each 5 seconds a new arrival. You know exactly at which time there will be an arrival. There is not any randomness at all. The letter M stands for memoryless or Markovian. This means that the time between two arrivals is independent and identically distributed, where the underlying distribution is an exponential distribution. We also speak of Poisson arrivals if the arrival pattern is specified with the letter M. We will discuss the exponential distribution later in more detail.

2. S - This letter specifies the service times. M, D and G are also in this case the most common letters. The meanings are the same. The letter G stands for a general distribution and thus we do not have any assumptions on the service times. The letter D stands for deterministic service times and thus each job needs exactly the same time. The letter M stands again for Memoryless or Markovian, which means that the service times follow an independent and identical exponential distribution.
3. c - This letter specifies the number of servers and is thus a number between 1 and infinity.
4. K - This letter specifies the maximum number of jobs in the system. This means the number of jobs in the queue and the number of jobs in service. This can be a finite or an infinite number, and is in general equal to or greater than slot c.
5. N - This letter specifies the calling population. There will be no more arrivals if the calling population becomes empty. The calling population can be finite or infinite. We will always assume in this thesis an infinite calling population.
6. D - This letter specifies the queueing discipline. This letter specifies the next job from the queue that will be served. Most common are first come, first served (FCFS), last come, first served (LCFS) and processor sharing (PS). FCFS means that all jobs will be served in order of arrival. If there is an empty server and there are more jobs in the queue, the job with the highest waiting time will be served. LCFS is the reverse of FCFS. Not the job with the highest waiting time, but the job with the lowest waiting will be served as first. PS is a totally different queueing discipline. One server handles all jobs at the same time, but it shares the workload over all jobs. We will discuss processor sharing later more extensively.

In some cases the last three slots are empty. In that case we assume an infinite maximum number of jobs in the system, an infinite calling population and the FCFS server discipline. I mentioned the  $M/M/1$  queue earlier. This means one queue with Poisson arrivals, one server with exponential service times, a infinite number of jobs in the system, an infinite calling population and a FCFS server discipline.

If we just speak of a global queueing system, we just say  $M/M/1$  for example. If we are speaking of a specific case, we also specify the variables. In the case of the  $M/M/1$  we have to choose the arrival rate and server rate, the parameter of the exponential distribution. The letter we give to the arrival rate is mostly the  $\lambda$  and the letter we give to the server rate is mostly  $\mu$ . So if we are talking about a  $M/M/1$  model with  $\lambda = 2$  and  $\mu = 3$ , we are actually talking about a  $M/M/1$  model where the time between two arrivals is exponentially distributed with mean  $\frac{1}{2}$ , and the service times are exponentially distributed as well with mean  $\frac{1}{3}$ . Figure 3.1 shows us a graphical notation of the  $M/M/1$



FIGURE 3.1: The  $M/M/1$  queue in graphical notation.

queue. We determine  $\lambda$  and  $\mu$  and put down a queue and a server. The  $M/M/c$  has almost the same graphical representation, but has more servers, and thus more circles. This will be the case for the  $M/M/1$  model. Another example is the  $M/M/1/n$  queue. Only the last two slots are empty in this case, and thus we assume an infinite calling population and the FCFS server discipline. If we only change the server discipline in the  $M/M/1$  queue, we will also denote the new system as  $M/M/1$ -PS for example if we use Processor sharing.

### 3.1.2 Exponential distribution

The exponential distribution is a special distribution from a mathematical point of view. It has some properties which makes our lives easier. The exponential distribution is closely related to the Poisson process, the minimum value of some exponentially distributed random variables is also exponentially distributed and the exponential distribution has the memoryless property. The probability density function of the exponential distribution with parameter  $\lambda$  is given by

$$f_{\lambda}(x) = \begin{cases} \lambda^{-\lambda x} & \text{if } x \geq 0; \\ 0 & \text{if } x < 0. \end{cases} \quad (3.1)$$

From here it follows that the cumulative distribution function is given by

$$P(X_\lambda < 1) = F_{X_\lambda}(a) = 1 - e^{-\lambda a} \text{ and } P(X_\lambda > 1) = 1 - F_{X_\lambda}(a) = e^{-\lambda a} \text{ for } a \geq 0. \quad (3.2)$$

The mean and variance of an exponentially distributed random variable  $X$  with parameter  $\lambda$  are given by

$$E[X] = \int_0^\infty x f_\lambda(x) dx = \frac{1}{\lambda} \text{ and } Var(X) = \int_0^\infty x^2 f_\lambda(x) dx - E[X]^2 = \frac{1}{\lambda^2}. \quad (3.3)$$

We will use this information to prove the following claims.

**Proposition 3.1.** *Let  $X = \text{Min}(X_1, \dots, X_n)$ . If  $X_1, \dots, X_n$  are independent exponentially distributed random variables with parameter  $\lambda_1, \dots, \lambda_n$  respectively, then  $X$  is also an exponentially distributed random variable with parameter  $\lambda = \sum_{i=1}^n \lambda_i$ .*

*Proof.*

$$P(X > a) = P(\text{Min}(X_1, \dots, X_n) > a) \text{ (by definition)} \quad (3.4)$$

$$= P(X_1 > a, \dots, X_n > a) \quad (3.5)$$

$$= \prod_{i=1}^n P(X_i > a) \text{ (by independence of the random variables)} \quad (3.6)$$

$$= \prod_{i=1}^n e^{-\lambda_i a} \quad (3.7)$$

$$= e^{-a \sum_{i=1}^n \lambda_i} \quad (3.8)$$

$$= e^{-a\lambda} \quad (3.9)$$

which proves the proposition.  $\square$

The use of this property may not be clear at this stage. The minimum value has something to do with the next event. We are interested in what the next event is and when will it happens. If we have some independent events which all follow an exponential distribution, we can apply this result. It turns out that this property combined with the memoryless property is very useful in queueing networks.

**Proposition 3.2.** *An exponentially distributed random variable  $X$  with parameter  $\lambda$  has the memoryless property. This means*

$$P(X > t + s \mid X > t) = P(X > s) \text{ for } s, t \geq 0 \quad (3.10)$$

*Proof.* This follows immediately from the definition of the conditional probability and the cumulative distribution function of the random variable.  $\square$



We can interpret this result as follows. Let us assume  $X$  is the time until the next event and we are already waiting for 10 seconds. We want to compute the probability that the next event will not occur after another 5 seconds. This conditional probability will be the same as the unconditional probability of waiting for more than 5 seconds. It does not depend on the current waiting time. The process simply does not remember the current waiting time. Therefore it is called memoryless.

### 3.1.3 Poisson process

**Definition 3.3** (Counting process). A stochastic process  $\{N(t), t \geq 0\}$  is a counting process if

1.  $N(t) \geq 0$ .
2.  $N(t)$  is integer valued.
3. If  $s < t$ , then  $N(s) \leq N(t)$ .
4. For any  $s < t$ ,  $N(t) - N(s)$  is equal to the number of events in the interval  $(s, t]$ .

**Definition 3.4** (Poisson process I). A counting process  $\{N(t), t \geq 0\}$  is a Poisson process with rate  $\lambda$  ( $\lambda \geq 0$ ) if

1.  $N(0) = 0$
2. The process has independent increments, i.e. for any  $0 < t_1 < t_2 \leq t_3 < t_4$  the increments  $N(t_2) - N(t_1)$  and  $N(t_4) - N(t_3)$  are independent.
3. The number of events in any interval of length  $t$  is Poisson distributed with mean  $\lambda t$ . I.e. for all  $t, s \geq 0$ , we have

$$P(N(t+s) - N(s) = n) = e^{-\lambda t} \frac{(\lambda t)^n}{n!} \text{ for } n = 0, 1, 2, \dots \quad (3.11)$$

**Definition 3.5** (Poisson process II). A counting process  $\{N(t), t \geq 0\}$  is a Poisson process with rate  $\lambda$  ( $\lambda \geq 0$ ) if

1.  $N(0) = 0$ .
2. The process has stationary and independent increments.
3.  $P[N(h) = 1] = \lambda h + o(h)$ ,  $P[N(h) \geq 2] = o(h)$  for small  $h$ .

**Theorem 3.6.** *Definition 3.3 and 3.4 are equivalent.*

*Proof.* We will first prove that definition 3.3 implies definition 3.4.

1.  $N(0) = 0$  holds for both definitions
2. The process has independent increments by (2) of definition 3.3 and the increments are stationary by (3).
- 3.

$$P[N(h) = 1] = \lambda h e^{-\lambda h} \text{ (by (3) of definition 3.3).} \quad (3.12)$$

$$= \lambda h(1 + o(h)) \text{ (using Taylor series)} \quad (3.13)$$

$$= \lambda h + o(h) \quad (3.14)$$

The proof for  $P[N(h) \geq 2] = o(h)$  is similar.

We will now show that definition 3.4 implies 3.3.

1.  $N(0) = 0$  holds for both definitions.
2. The process has independent increments by (2) of definition 3.4.
3. Let us define  $g(t) = E[e^{-uN(t)}]$  for  $u \geq 0$ . We can derive the following differential equation for  $g(t)$ :

$$g(t+h) = E[e^{-uN(t+h)}] \quad (3.15)$$

$$= E[e^{-uN(t)} e^{-u(N(t+h)-n(t))}] \quad (3.16)$$

$$= E[e^{-uN(t)}] E[e^{-u(N(t+h)-n(t))}] \quad (3.17)$$

$$\text{(by independent increments)} \quad (3.18)$$

$$= g(t) E[e^{-uN(h)}] \text{ (by stationary increments).} \quad (3.19)$$

We are able to calculate  $E[e^{-uN(h)}]$  with (3) of definition 3.4.

$$E[e^{-uN(h)}] = (1 - \lambda h + o(h)) * 1 + (\lambda h + o(h))e^{-u} + o(h)e^{-2u} \quad (3.20)$$

$$= 1 - \lambda h + \lambda h e^{-u}. \quad (3.21)$$

This leads us to the expression  $g(t+h) = g(t)(1 - \lambda h + \lambda h e^{-u})$  and thus

$$\frac{g(t+h) - g(t)}{h} = g(t)\lambda(e^{-u} - 1) + \frac{o(h)}{h}. \quad (3.22)$$

Letting  $h \rightarrow 0$  gives us  $g'(t) = g(t)\lambda(e^{-u} - 1)$ . This expression with the starting condition  $g(0) = 1$  give a differential system with solution  $g(t) = e^{\lambda t(e^{-u}-1)}$ . This

expression is equal to the Laplace-Stieltjes transform of a Poisson random variable with mean  $\lambda t$ . Since each non negative random variable is uniquely determined by its Laplace-Stieltjes transform, we can conclude that  $N(t)$  is a Poisson random variable.

□

We looked at two different definitions of the Poisson process since the first definition is intuitively more clear, but the second definition is more handy to prove some properties of the Poisson process. The two properties we will discuss are that splitting and merging a Poisson process results in a new Poisson process.

**Proposition 3.7.** *Let  $\{N_1(t), t \geq 0\}$  and  $\{N_2(t), t \geq 0\}$  be two independent Poisson processes with rate  $\lambda_1$  and  $\lambda_2$ . Then the process  $N(t) = N_1(t) + N_2(t)$  is also a Poisson process with rate  $\lambda_1 + \lambda_2$ .*

*Proof.* Checking the conditions in definition 3.4 for  $N(t)$  proves this proposition. □

**Proposition 3.8.** *Let  $\{N(t), t \geq 0\}$  be a Poisson process with arrival rate  $\lambda$  and let us mark each arrival with probability  $p$ . Denote  $N_1(t)$  as the number of marked arrivals and  $N_2(t)$  as the number of not marked arrivals. Then  $N_1(t)$  is a Poisson process with rate  $p\lambda$  and  $N_2(t)$  is a Poisson process with rate  $(1 - p)\lambda$ .*

*Proof.* Checking the conditions in definition 3.4 for  $N_1(t)$  and  $N_2(t)$  proves this proposition. □

These two propositions will be very handy later on. We will use these properties of the Poisson process to analyze our analytical model for the CreateOrder service. It is very handy to know that the splitted or merged process is again a Poisson process.

### 3.1.4 PASTA

Another important property of a Poisson process is the PASTA property. This is the abbreviation for Poisson arrivals see time averages. This property goes about M/./ queueing systems. This makes analyzing a system with Poisson arrivals easier than analyzing other systems. A new arrival sees the average behavior of the system, which is not always the case. Let us have a look at a D/D/1 system where there is an arrival at  $t = 2, 4, 6, 8, \dots$  and with the deterministic server time of 1 time unit. Each customer arrives at a empty system. If the PASTA property was true for this system, we could

conclude that the system is always empty. But this is of course not the case. The system is empty half of the time and busy half of the time. This shows that the PASTA property does not hold for this system.

The following is true for M/./ queueing systems: Let  $p_n$  be the fraction of time that there are  $n$  costumers in the system. Then the probability that there are  $n$  jobs in the system if a new job arrives is equal to  $p_n$ . The fraction of the customers who arrive if there are  $n$  customers in the system is equal to the fraction of the time the system spends in the state with  $x$  customers.

### 3.1.5 (Limited) Processor Sharing

The server discipline used in the system of the ING Bank is processor sharing. A server with this discipline can handle more than a job at a time, but it will work a lot slower. An M/M/1 queueing system with arrival rate  $\lambda$  and mean service time  $\mu$ , can handle 1 job at a time at a rate of  $\mu$ . An M/M/1-PS queueing system with the same parameters handles all jobs at the same time. But, given that there are  $k$  jobs in the system, the server rate per job is  $\frac{\mu}{k}$ . This means that the work speed of a server depends on the number of jobs that are in services. An arrival or departure changes the server rate. So the total server rate will still be  $\mu$ , but the server shares the rate over all the jobs.

Limited processor sharing is a discipline between processor sharing and FCFS, although it looks like most processor sharing. By processor sharing the server can handle any number of jobs. By limited processor sharing, this number is limited by some  $n$ , the other jobs are waiting in the queue. That is the difference between processor sharing and limited processor sharing. The server rate of a single job is equal to  $\frac{\mu}{k}$  if there are  $k \leq n$  jobs in the system and  $\frac{\mu}{n}$  if there are more than  $n$  jobs in the system. If  $n = 1$  we actually have a FCFS queue, since the server can handle only 1 job with full speed.

### 3.1.6 Sessions

In certain queueing networks we will also use the concept of sessions. A session limits the number of jobs served in that part of the queueing network. If we speak about 15 sessions, that means that there will be at most 15 jobs in that part of the queueing network. So it can be the case that a job enters the queue of a server which can handle one more job, but all sessions are busy. In that case the job cannot be served and has to wait until there is a free session. Let us have a look at an example. Figure 3.2 shows us a simple queueing network of two M/M/c queues in series with sessions from the first server until the last server. Let us say there are 5 sessions. It is possible that at some moment there is one job in service at the second server and there are 4 jobs waiting in

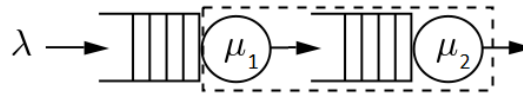


FIGURE 3.2: Graphical notation of sessions: a rectangle with dotted line. The sessions start when the job can be served at the first server and ends after the service of the second server

the second queue. These are all the jobs in the system. At some moment, a new job arrives. The first server is idle, but is not allowed to serve the new job, since all sessions are busy. Therefore, the new job has to wait in the queue.

This example shows that sessions have a negative influence on the performance of the system. A new job cannot be served immediately and has to wait, so the total time spent in the system becomes a bit longer. Completely deleting the sessions or increasing the number of sessions will often increase the performance of the system.

### 3.1.7 Time to live

Time to live is a mechanism used in many computer networks. It limits the life time of a job in the network. In some cases a job can be stuck in a queueing network. By bad luck each step takes a really long time. To eliminate these scenarios, we applied the concept of time to live. A job has a certain life time. All jobs which exceed this life time will be eliminated. We can apply the time to live on a single server, a single queue or greater parts of a queueing network, including the whole system.

There are also other options than eliminating the job completely. If the time to live is applied on a part of the queueing network, we can also decide that the job skips only that part of the queueing network and continues as if this specific part is finished. Or we can define a completely different flow for these jobs. So there are many options.

It is also possible to have some nested life times for jobs. We can for example have a queueing network with time to live applied on the whole system and on each server. A job will be dropped if one the different times to live is exceeded. Or and how the job continues after that moments depends on which life time is exceeded.

### 3.1.8 Important measures

In queueing theory there are some important measures which we are interested in. We will discuss some of these measures in this section.

- $E[L]$  is the long-run average number of customers in the system.

- $E[L_q]$  is the long-run average number of customers in the queue.
- $E[S]$  is the long-run average sojourn time of a customer in the system. This is equal to the waiting time in the queue and the service time.
- $E[W_q]$  is the long-run average waiting time of a customer in the queue.

### 3.1.9 Little's law

Little's law [2, 3] is a law which links the average number of customers, the average waiting time and the arrival rate. Let  $L$  be the average number of customers in the system, let  $W$  be the average waiting time and let  $\lambda$  be the arrival rate. Then Little's law is given by

$$L = \lambda W \quad (3.23)$$

if the process is in steady state distribution. Note that we do not include any assumption on the arrival process or the service time distribution or number of servers.

We can apply this law on the complete queueing network of a queue and get  $E[L] = \lambda E[S]$ . If we apply the law on a queue we get  $E[L_q] = \lambda E[W_q]$ . This is very useful. Mostly, two of the three parameters are easy to determine, but the third is difficult, using Little's law, we can easily determine the third unknown.

Let us have a closer look at why Little's law is true. Let us define

- $n(t)$  = the number of jobs in the system at time  $t$ ,
- $T = [t_1, t_2]$  be a large interval,
- $A(T)$  be the surface under the curve of  $n(t)$  and
- $N(T)$  be the number of arrivals in interval  $T$ .

Let  $L(T)$  be the average number of customers in the interval  $T$ . Then  $L(T) = \frac{1}{T} \int_{t_1}^{t_2} n(t) dt = \frac{A(T)}{T}$ . The arrival rate in the interval  $T$  is given by  $\lambda(T) = \frac{N(T)}{T}$ . The total waiting time for the jobs in the interval  $T$  is given by  $\int_{t_1}^{t_2} n(t) dt = A(T)$ . Therefore the average waiting time of a single job is  $W(T) = \frac{A(T)}{N(T)}$ .

From here it follows that  $L(T) = \lambda(T)W(T)$ . Under reasonable assumptions about the steady state distribution it follows that  $L(T)$  converges to  $L$ ,  $\lambda(T)$  converges to  $\lambda$  and  $W(T)$  converges to  $W$  if  $T$  goes to infinity and we get  $L = \lambda W$ .

## Chapter 4

# Simulation

In our first approach we will use the technique of simulation to solve our chosen model. The easiest way to explain a simulation is to give an example. We will use the M/M/1 model as example. The simulation generates jobs according to a given arrival process. If the server is free, it assigns the first job to it and determines a service time based on the server time distribution. At the same time the simulation keeps track of some statistics. It calculates for example the mean server time, the mean waiting time, the mean number of jobs in the system, the utilization of the server, etc. After a certain amount of time (or until some termination condition) the simulation stops and outputs the statistics.

We already introduced the M/M/1 model. Normally we have to analyze the real life system and make the translation to a mathematical model. We used the M/M/1 model, but actually we assume that the M/M/1 model imitates the real life process very well. Running a simulation starts always by a real life process which must be translated to a mathematical model. This translation can be very complicated.

Now we have validated the mathematical model and want to run the simulation. This brings us some questions.

- How many runs?
- How long does a run last?
- How long does the warm-up period take?

Based on theorems like the Central limit theorem and the Law of large numbers we assume our statistics converge to the real value if our running times goes to infinity. Simulating forever is not an option. Therefore we will have faith in long runs, but we do not have the guarantee that the value we obtain is really close to the real value.

Therefore we perform several runs.

You also have to take care of the warm-up period. In queueing networks you mostly start with an empty system. This has influence on the performance later on. Another starting point would lead to different short term results. Therefore we will use a warm-up period and we assume that the process is in steady state behavior after this period. We do not gather the results of the warm-up period.

The best way to obtain the answers to the questions we posed is to run the simulation several times, but with different settings, changing the warm-up period a bit. If the simulation runs long enough, we believe that the simulation is in steady state behavior. The warm-up period has to be chosen such that the influence of the starting conditions are negligible.

## 4.1 Rockwell Arena

There are many tools that can be used in order to simulate a given model. In this thesis we will use the tool Rockwell Arena. Before we look at the CreateOrder, we first introduce Rockwell Arena. Rockwell Arena is a visual simulation tool which is suited for simulating queueing systems. This program provides its own help documentation that explains well the basics of the program. Therefore we will only give a quick review of the boxes that we used in the modeling of the LOM.






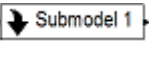




Rockwell Arena can be seen as a drag and drop tool. You can construct queueing networks by dragging and dropping some boxes and determining a flow through the boxes by simply drawing some lines between the boxes. There are many different boxes with their own functionality, there is for example a box to determine the arrival pattern. Table 4.1 gives a brief explanation of the most common boxes. There are other different boxes, but we will not need them for modeling the CreateOrder.

By using some boxes and lines we can construct a queueing network. A flow always starts in a create box and ends in a dispose box. It is possible to use boxes multiple times. To follow the flow we will use the following rule: All incoming requests to a box enter the box from the left. All outgoing requests leave the boxes from the right. Based on this, we can easily determine the flow in Rockwell Arena.

We will illustrate this with some examples in the remaining part of this section. Our first example will be the M/M/1 model. After our first model we will expand the M/M/1 model with some concepts which we also use in our model of the CreateOrder service. Another way to learn Rockwell Arena is by reading the help documentation of Rockwell Arena [4] providing also a tutorial of the basics.



TABLE 4.1: Arena

| Name   | Explanation   |
|--|---|
| <br>Create      | In this box we can specify the arrival pattern. We will have many choices: The arrival pattern can be deterministic or stochastic. The arrivals can be single jobs or in batches.   |
| <br>Dispose     | A job entering this box leaves the system.  |
| <br>Decide      | Based on some conditions or on some probability law the flow will be divided in two parts.  |
| <br>Hold        | This box is used to hold a job for a certain time. It can be held until a certain condition is fulfilled or until a certain signal is given.  |
| <br>Signal      | A job entering this box gives a signal to a certain hold-box, so the hold-box releases all jobs.  |
| <br>Submodel 1 | This box is used to shrink multiple boxes into one box. It can be used to give the general flow of a complicated process. There can be some difficult processes in the submodel box. To open the submodel just double click on the box. |
| <br>Assign    | This box is used to assign variables or attributes and so on.   |
| <br>Record    | This box is used to gather some statistics.   |
| <br>Delay     | This box delays the job for a certain amount of time. This can be deterministic or stochastic.  |
| <br>Process   | In this box you can specify the processing time of the job. This can be deterministic or stochastic. You can also specify the number of services and so on. This box also contains a queue in case all servers are busy.                |

#### 4.1.1 A M/M/1 model in Arena

We already introduced the M/M/1 model with parameters  $\lambda$  and  $\mu$ . We assume that the values of  $\lambda$  and  $\mu$  are known. For example:  $\lambda = 1$  and  $\mu = 2$ . This model is easy to build in Rockwell Arena. It will use only three boxes: A create box, a process box and a dispose box. In the create box we define the arrival pattern. In this case the time between two arrivals is exponentially distributed with mean  $\frac{1}{\lambda}$ . In the process box we define the mean server time. This will be of course an exponential distribution with mean  $\frac{1}{\mu}$ . We also define one server. Within the process box we can also define the server

discipline. If no specification is given, Rockwell Arena chooses for the FCFS discipline. We do nothing within the dispose box. Figure 4.1 shows the M/M/1 model in Rockwell Arena.

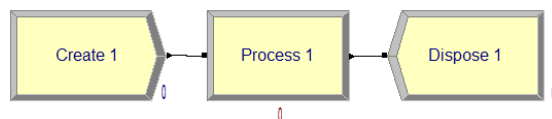


FIGURE 4.1: M/M/1 in Arena

We can easily expand this queueing system to another queueing system. We define the arrival pattern in the create box. We can easily choose another distribution than the exponential distribution. This gives us the opportunity to create a G/M/1 queue. In the process box we define the server time distribution. Instead of our exponential distribution we can also choose another distribution to create a M/G/1. We can also change the server discipline. But here we have some troubles: Rockwell Arena knows only three types of disciplines: FCFS, LCFS and a random order. So we can only change the server discipline to one of these choices. We have to find a solution for other server disciplines, like processor sharing. Since the solution is not that easy, we will first have a look at other models.

### 4.1.2 Gather statistics

After running a simulation, Rockwell Arena gathers a lot of statistics by itself. These statistics are mostly useful for simple queueing networks, for example the M/M/1. Rockwell Arena provides statistics over the number of jobs in the system, the number of jobs in the queue, the waiting time, the server time, etc. It gathers statistics like the mean, the minimum, the maximum, etc. But these are not always the statistics we are looking for. We can also define our own statistics in Arena with corresponding assign and record boxes.

An example of such statistic, could be the total drop off of some server. We only need a record box at the drop off point. One option in the record box is to count the number of jobs through the box. This is exactly the statistic we were interested in.

Another statistic is the time spent in a specific part of the queueing network. To keep track of this time we need an assign box at the start of the part we are interested in. The assign box gives each job an attribute with the current time as value. The record box is at the end of the specific part. Here we choose the option time interval and then we choose the earlier specified attribute.

### 4.1.3 Maximum queue length in Arena

In the section we will expand the M/M/1 with a finite queue length. In fact we will try to create a M/M/1/n queue. The maximum queue length is unfortunately not a parameter in Rockwell Arena. Therefore we have to use some boxes in a clever way so the system behaves like a system with a finite queue length. This can be done by using a decision box.

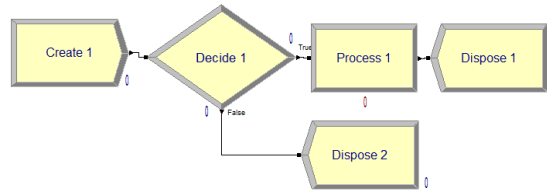


FIGURE 4.2: M/M/1/n in Arena

Figure 4.2 shows our solution. We started with the M/M/1 queue and used a decision box to keep track of the numbers in the queueing. The decision box will determine if the queue is full or not and may or may not give the job the permission to enter the queue. If the decision box sees a full queue, the job will be blocked and immediately leaves the system. We have earlier seen that we can expand our M/M/1 easily to a G/G/c. We can do the same for the M/M/1/n.

### 4.1.4 (Limited) Processor sharing in Arena

We also have to find a way to model processor sharing. Rockwell Arena does not have this server discipline. The problem is that there is not any easy solution. The problem is even bigger: Processor sharing cannot be constructed in Rockwell Arena. Our only option is a good approximation which is not time consuming. Our idea is to keep track of the remaining server times and update these regularly. If the time between two updates converges to zero, the constructed process converges to processor sharing. On the other hand the running time of Rockwell Arena increases a lot if we decrease the time between two updates. Therefore we have to find a good balance, between small enough to be a good approximation, but not too small, since it will be otherwise too time consuming.

Figure 4.3 shows our model for the M/M/1-PS queue. The system consists of three parts. The upper part is the queue itself. The lower part is used to define the time between two updates of the processing time. Let us have a closer look at the upper part. Jobs arrive in the arrival box and get a total processing time in the second box. This is the time the job will spend in the system if there are no other jobs in the system.

The third box is also used to give a signal to update the work left for each job. This is

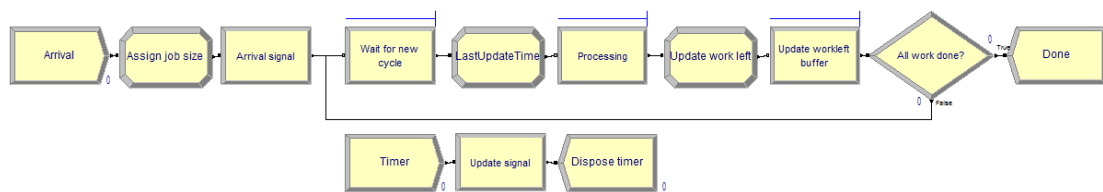


FIGURE 4.3: Processor sharing in Arena

a good moment to update the work left, since the work speed of the server on each job will change.

The remaining part of the model is used to update the work left for the jobs. The jobs wait in the "Wait for new cycle"-box until all work left is updated. In the next box, the current time is assigned to the job, so it is known when the job starts processing in the next cycle in the processing box. Based on the time spent in the processing box and on the number of jobs in the system the work left is updated in Update Work Left. Jobs have to wait in the Update workleft buffer box until all work left is updated. At the last box of the cycle, the "All work done?"-box, it checks if the work left is equal to or less than zero. If that is true, the jobs leave the system, if that is false, the cycle starts again.

| $\lambda$ |                 | 10     | 1     | 0.1   | 0.01  | Theoretic |
|-----------|-----------------|--------|-------|-------|-------|-----------|
| 0.1       | Processing time | 6.446  | 1.620 | 1.099 | 1.052 | 1.111     |
| 0.3       | Processing time | 7.028  | 2.168 | 1.442 | 1.373 | 1.429     |
| 0.5       | Processing time | 7.892  | 2.875 | 1.872 | 1.769 | 2         |
| 0.7       | Processing time | 10.242 | 3.794 | 2.518 | 2.378 | 3.333     |
| 0.9       | Processing time | 11.042 | 4.514 | 2.962 | 2.792 | 10        |

TABLE 4.2: Our approach for a M/M/1-PS queue in Rockwell Arena under different step sizes (from 10 to 0.01) compared to the theoretic result.  $\mu = 1$ , different values for  $\lambda$ .

There is only one problem: Rockwell Arena cannot handle these kind of loops very well. Table 4.2 shows us some values for different inputs. A smaller step size must lead to a smaller difference with the theoretic value. This table shows us the failure of this approach. The failure has to do with the asynchronous way of updating the jobs in Rockwell Arena. To fix this problem we have to write some java file which gives Rockwell Arena the task to handle the jobs synchronously. This makes Rockwell Arena very slow and therefore it is not recommended to use this approach.

Therefore we have to come up with another idea. We use our knowledge of queueing

theory. The M/M/1 queue and M/M/1 processor sharing queue have a lot in common. The total number of jobs in the system is in both cases the same. The same holds for the mean time spent in the system. We can prove this claim with the Markov theory, which we will discuss in Chapter 5. Using a M/M/1 queue instead of the M/M/1 processor sharing queue is therefore the best approximation of a M/M/1 processor sharing queue. The difference between processor sharing and limited processor sharing is not big. We will face the same problem as for processor sharing. It cannot be done in Rockwell Arena. The behavior of processor sharing and limited processor sharing is almost the same. Instead of modeling the M/M/1 limited processor sharing queue we will model the M/M/1 queue. We will also prove in Chapter 5 that limited processor sharing has the same mean time spent in the system and the mean number of jobs in the system. Our best approximation of the M/M/1-PS queue is thus the M/M/1 queue. They have a lot of similarities in mean values analysis, although the underlying distributions of the waiting time, the server time, etc. are completely different. Aspects of the number of parallel servers and the mean server time are much more important. In our analysis of the CreateOrder service we will mainly focus on the influence of these two aspects of the queueing system and not on the number of jobs a server can handle at the same time.

#### 4.1.5 Sessions in Arena

Adding sessions to the model is not that difficult. At the start and the end of the session, we use assign boxes to keep track of the number of used sessions. Based on where the session starts, we can add a hold box before the start of the session, we construct a queue.

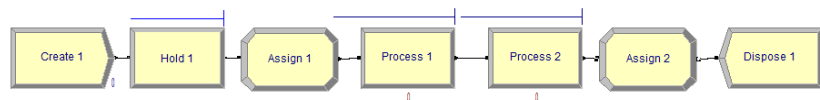


FIGURE 4.4: Sessions in Arena. The sessions start when the job can be served at the first server and ends after the service of the second server. The hold box is considered as a queue.

#### 4.1.6 Time to live in Arena

Time to live can be applied on different parts of the queueing system. It can be applied on the queue, on the server, on the whole system, . . . But Rockwell Arena does not have this functionality. We have to create it ourselves. Let us first look at a M/M/1 queueing system with Time to Live applied to the queue. The job that spends the most time in the queue is the next job which will be served. Instead of checking the waiting time for

each job we can only check the waiting time for only that specific job. Unfortunately, Rockwell Arena cannot check this in continuous time. We have to make our own approximation.

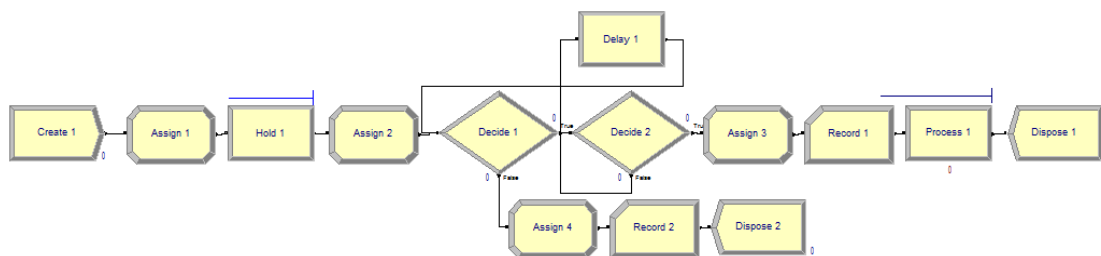


FIGURE 4.5: Time to live applies on a queue

We assume that the job with the highest waiting time is in front of the queue. This means we know on which job the "time to live" will be exceeded first. We create a loop with just one job. The job with the current highest waiting time. We fix also some small time interval. At the end of each time interval we will check the time to live. If the life time is exceeded, the job will leave the queue. Otherwise it will continue. Then it checks if the job can be in service. If that is the case, the job enters the server and a new job enters the loop. Otherwise the job will still be in the loop for the same amount of time. Figure 4.5 shows us our approximation of the time to live applied on a queue. In this figure are also some record boxes. The record boxes record the time that a job spends in the system from box "Assign 1" until the record box. Let us assume we have a time to live of 5 seconds. In box Record 1 all jobs have a service time less the 5 seconds. Note that our approach was an approximation. If our approach was exact, all times recorded in box 2 will be exactly 5 seconds. Now it is a little bit higher. At most the delay we have to choose. Since this construction also has a loop, and Arena can not handle loops very well, we will try to avoid Time to live applied on a queue. But this is not always possible.

We can also extend this model with the ideas we already discussed. We could change the server discipline to processor sharing or we can change the infinite queue to a finite queue. It is also possible to apply this concept to a G/G/c queue.

Our next goal is to look at a queueing system with time to live on the servers. This can be done easily. Let us have a look at a M/M/1 queue with a time to live of  $x$  time units. Instead of having Expo(mean) as server time distribution in Rockwell Arena, we will use mn( $x$ , Expo(mean)). The mn is the command for minimum in Rockwell Arena. This works as follow: A job enters the server and gets some work done according some exponential distribution. If this work is more than the time to live, it is limited to the time to live. We also add a decision box to the model, right after the processing box.

We will check in this box why the job leaves the server. Is the work done (service time less than  $x$ ) or is the time to live exceeded (service time equal to  $x$ ).

Time to live does not have to be applied to a single queue or server only, it could also be applied on a bigger part of a queueing network. Our approach can still work. Let us have a look at a M/M/1 queue with time to live on the whole process. We can apply our approach to the queue and server, but we must be careful. We had a reasoning in our approach for the time to live applied on a queue which may not hold in this case. Since the queue is large enough, there will be no blocking. Therefore, if we apply only a Time to live check on the server, it could work. If the waiting time is already larger than the time to live, the server time will be 0 and thus will not block the server. We can use a decision box to keep track of the number of jobs with no drop off and calculate the mean sojourn time of these jobs. Blocked jobs will get a sojourn time that is equal to the time to live.

## 4.2 CreateOrder in Arena

In the previous section we have developed some tools to construct some parts of a queueing system. Now it is time to create the CreateOrder in Rockwell Arena. Therefore we have to glue the ideas we discussed earlier together. Sessions, maximum queue lengths, processor sharing, time to live, they are all used in the CreateOrder flow.

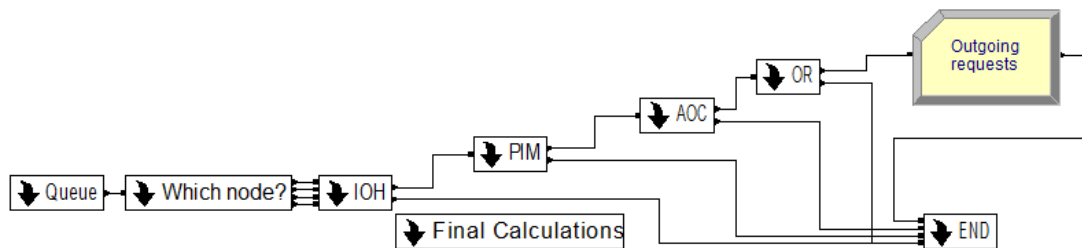


FIGURE 4.6: An overview of the complete model.

### 4.2.1 Arrival pattern and queue

Figure 4.7 shows us the arrival process and the queue of the LOM. The arrival process is just an arrival box. Within this box we have to specify the arrival distribution. We will choose for Poisson arrivals and include a variable to the model for the arrival rate. Later on we will simulate the model under different arrival rates to see how the system behaves under different workloads and to obtain the bottlenecks of the system.

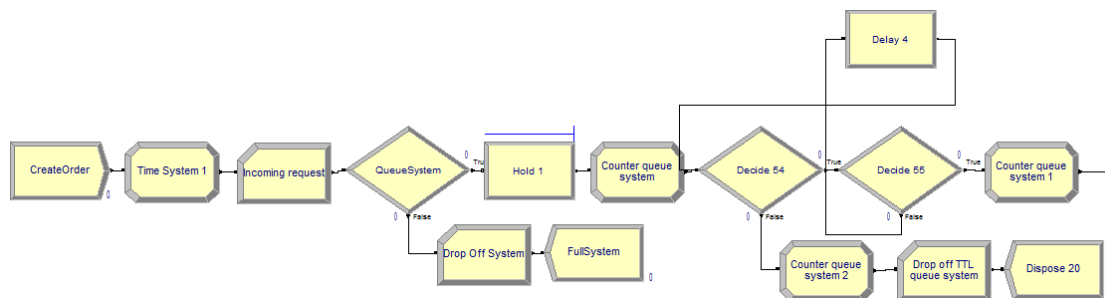


FIGURE 4.7: The arrival pattern and the queue

All the arrivals are put on the same queue, since the LOM has only one incoming queue. The hold box holds the jobs until there is a node in which there is a free listener and a free node. The queue has a maximum queue length and a time to live. If a new job sees a full queue, the job immediately leaves the system.

We also include a time to live check on this queue. Normally it is the customer who defines the life time of a the job, but we will assume a fixed number for each job. This does not influence the performance of the network. There is a blocking only if the system is unstable and the queue will grow to infinity (if possible). With the time to live a job is blocked because the life time is exceeded. Without the time to live, almost the same number of jobs are blocked, since the queue becomes full.

### 4.2.2 Splitting

The CreateOrder service has one queue, but is processed in four parallel nodes. Therefore all jobs are divided over the four nodes. We will call the step 'splitting'. We use decision boxes to determine the node in which the job will be executed. Jobs enters this splitting only if there is a node with a free listener and a free thread. If there are more nodes with a free listener and free thread it will choose the node with the least number of used listeners.

### 4.2.3 IOH 1

The node is chosen and the job starts with processing. First the job claims a thread and listener. The thread and listener will be occupied until the end of the complete process. In other words: A session starts at the start of the processing and finishes when the complete task is finished. Now the session is started, the job will be processed at the Initiate Order Handler of the specific node.

The server discipline of the nodes is processor sharing. Normally we had to use our approximation for processor sharing four times. With a little trick we only have to use



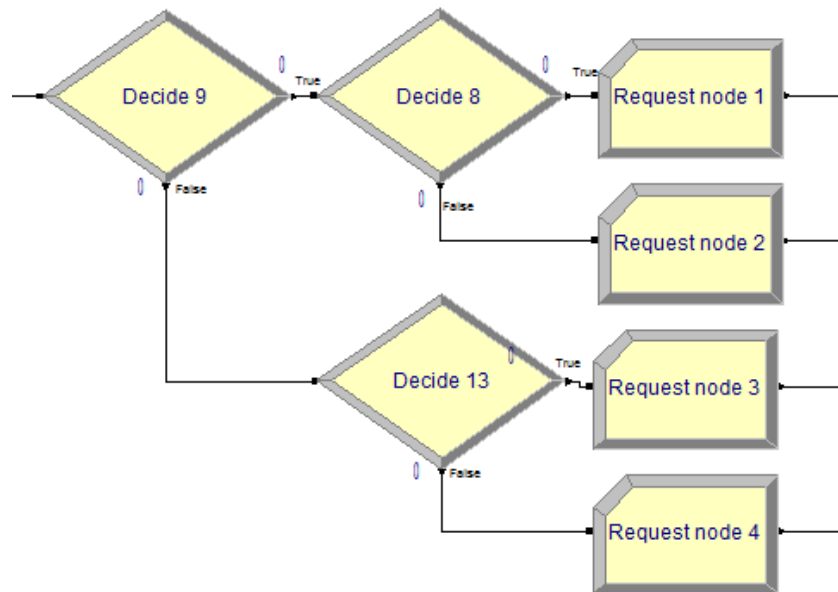


FIGURE 4.8: This figures shows the boxes which are used to determine the node in which the job will be executed.

it once. The loop in each four cases will be the same. The only difference is in the work left update, since it depends only on the number of jobs in the same node of the job. With some indicator function we can glue the work left updates together. This will do the trick.

We assume that the processor time of the different nodes are independent. The jobs in other nodes do not influence the processor time of the initiate order handler. We also assume exponential processor times. Under these assumptions, the distribution we have has one parameter: The mean processor time. These are known in practice.

Each session has a time to live. Within the loop we also include a time to life check. If the time to live expires, the job will leave the system. Therefore the job continues to the last part of the system to release the listener and thread.

#### 4.2.4 PIM

There is only one PIM. Therefore all jobs from the IOH go to the same queue. This is a processor sharing queue. Therefore we have modeled it as  $M/M/1$ , as discussed earlier. There is a limited queue length and there is time to live applied on the queue and server of PIM. Therefore we will keep track of the time spent in this part of the system and add a time to live check at the server. Since there is also a time to live check on the session, we have to decide why a job is blocked at this server: Is it from the time to live of PIM or is it because the time to live on the sessions. The remaining process is different in both cases. If the time to live on the session is exceeded, the process continues to the

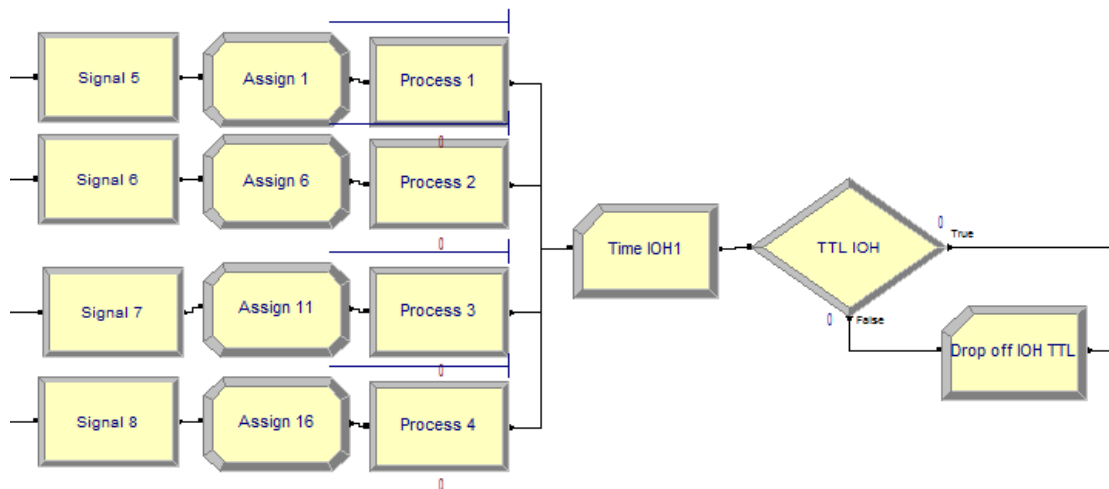


FIGURE 4.9: Start of a session and processing a job in the Initiate Order Handler

last step: The end of the sessions and leaves the system. If the time to live of PIM is exceeded, it just continues to AOC. Figure 4.10 shows how we modeled the PIM.

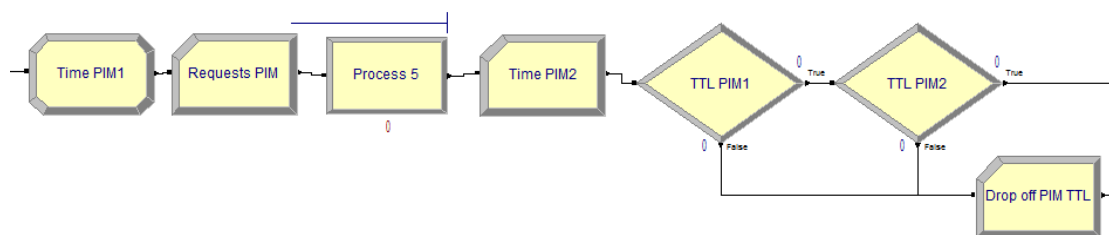


FIGURE 4.10: PIM

#### 4.2.5 IOH 2

The next step of the CreateOrder is again the IOH. The IOH receives the message of the PIM and sends immediately a request to AOC. The time that the job spends in this step is negligible. We could include this step in our model, but it has no influence on the bottleneck. Therefore we will not include this step in our model.

#### 4.2.6 AOC

Therefore the job continues to the AOC. From a mathematical point of view this machine is almost identical to PIM, with only other parameters. Processor sharing, time to live, limiting queue length, ... If the time to live of AOC exceeded, it continues to OR. If

the time to live of the session if exceeded, the job leaves the system and a new session becomes available. Figure 4.11 shows how we modeled this part of the queuing network.

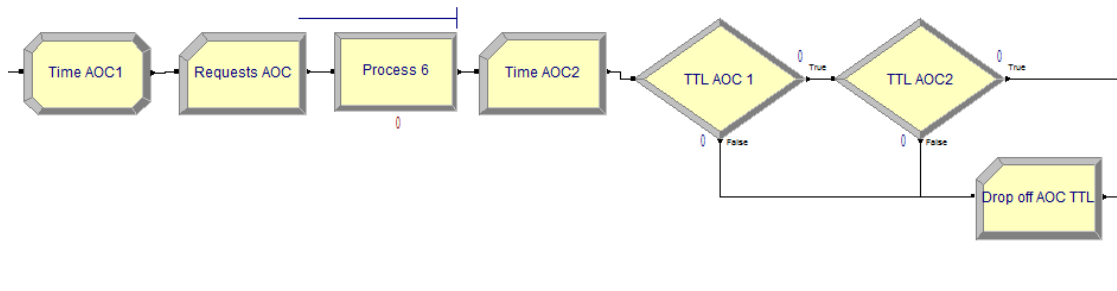


FIGURE 4.11: AOC

### 4.2.7 IOH 3

After the AOC the jobs go again to the IOH. The same as for IOH 2 holds: The time that a job spends in this step is negligible. The listener receives the message from AOC and sends immediately a request to OR. Therefore we do not include this step in our model.

### 4.2.8 OR

The OR is deployed on 4 nodes. A job goes to the same node as in the case of the IOH. Therefore the splitting is based on an earlier defined attribute, the attribute which specified the node of the IOH. With some decision boxes, a job goes to the right server. Each server has the server discipline limited processor sharing and has a finite queue length and is thus modeled as M/M/1/n queue.

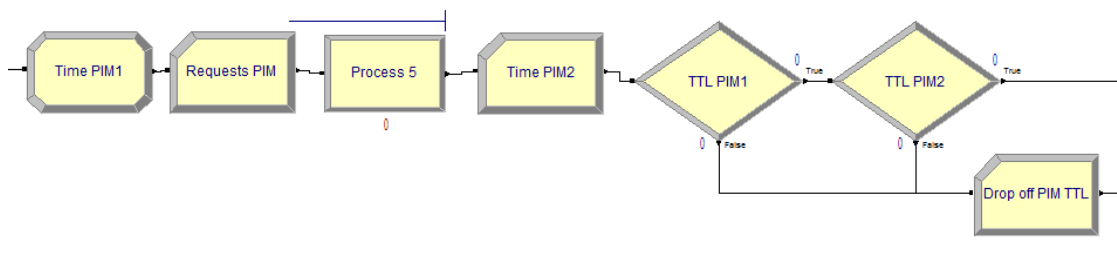


FIGURE 4.12: OR

### 4.2.9 IOH 4

After OR the jobs go back to IOH. The IOH sends now a reply to the customer and the task is done. The time spent in this step is again negligible. Therefore we do not

include this step. It is used to end the sessions. Therefore the job has to go to the right node, to end the session. This splitting is the same as for OR. It is done again, since some drop-offs must split at this moment. Figure 4.13 shows how it is done.

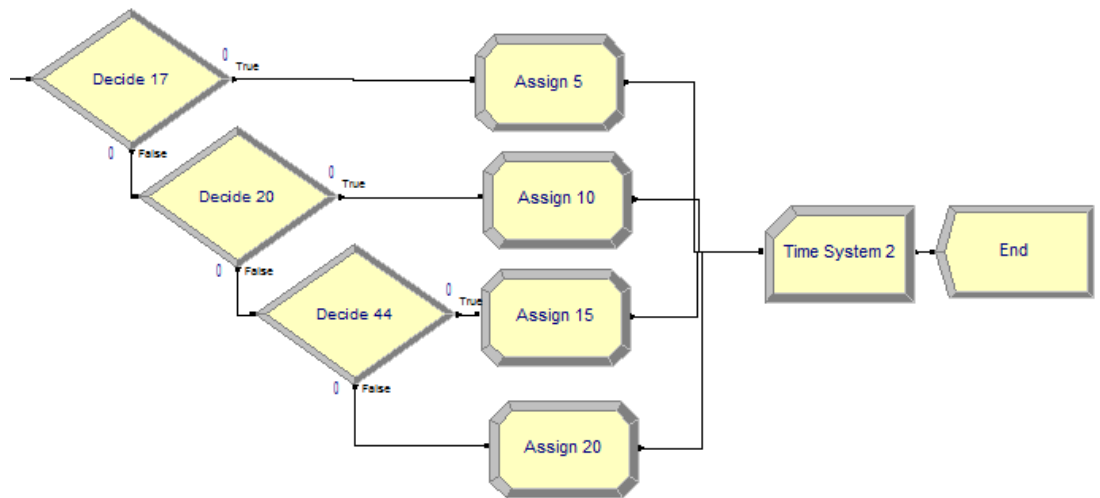


FIGURE 4.13: Session ending

#### 4.2.10 Variables

In order to adjust the behavior of the system, we need some variables in the system. Some of these variables are used to let the process run smoothly. Other variables define parameters in order to change the behavior of the system. The next tables will explain the variables. We will also include some tables to explain the recorded statistics which will be used to analyze the system.

Table 4.3 shows us a lot of variables which may not be adjusted. They are used to let the program run well. These parameters keep for example track of the numbers of sessions during a run. `#listener1` is for example used to keep track of the used number of listeners at node 1. This variable can not be higher than some number, since all the sessions at the first node will be used at that moment.

Table 4.4 shows us a lot of parameters to adjust the behavior of the system. It defines the server times, the queue lengths, the arrival pattern. When we analyze this system, we will change some of these parameters to see how the system behave under the changes.

We also introduce some tables with recorded statistics from a simulation. Table 4.5 shows a table with some tally statistics. In most cases these statistics are used to calculate statistics of time interval, but the first statistic is some expression, calculated just before the process ended.

| Variable            | Meaning   | Adjust variable? |
|---------------------|---|------------------|
| #AOC                | Number of jobs in service at AOC                                      | No               |
| #Counterqueuesystem | Used to let Time to live applied on the incoming queue works properly | No               |
| #Counterqueuepim    | Used to let Time to live applied on the queue of PIM works properly   | No               |
| #Counterqueueaoc    | Used to let Time to live applied on the queue of AOC works properly   | No               |
| #listener1          | Number of used listeners at node 1 at each moment in time             | No               |
| #listener2          | Number of used listeners at node 2 at each moment in time             | No               |
| #listener3          | Number of used listeners at node 3 at each moment in time             | No               |
| #listener4          | Number of used listeners at node 4 at each moment in time             | No               |
| #OR1                | Number of jobs in service at OR in node 1                             | No               |
| #OR2                | Number of jobs in service at OR in node 2                             | No               |
| #OR3                | Number of jobs in service at OR in node 3                             | No               |
| #OR4                | Number of jobs in service at OR in node 4                             | No               |
| #PIM                | Number of jobs in service at PIM                                      | No               |
| #thread1            | Number of used threads at node 1 at each moment in time               | No               |
| #thread2            | Number of used threads at node 2 at each moment in time               | No               |
| #thread3            | Number of used threads at node 3 at each moment in time               | No               |
| #thread4            | Number of used threads at node 4 at each moment in time               | No               |

TABLE 4.3: Variables that let the process run smoothly

Table 4.6 shows another set of variables. These are counting statistics and count the number of jobs that go through some part of the system. It calculates the number of incoming jobs, the number of outgoing jobs, the number of drop offs at each stage of the queueing network, ... These expressions are useful to analyze the performance of the system.

### 4.3 Bottleneck analysis

We now analyze the model we just described. This is done by simulating the model under different loads and different scenarios. The first important step is determining the right replication length and warm-up period. This is usually done by simulating a system with the same parameters for different times and compare the results. If the

| Variable          | Meaning   | Adjust variable? |
|-------------------|---|------------------|
| ArrivalPatternTPS | Average number of arrival per second                    | Yes              |
| MeanAOC           | Mean server time of a single job at AOC                 | Yes              |
| MeanListener1     | Mean server time of a single job at IOH1                | Yes              |
| MeanOR            | Mean server time of a single job at OR                  | Yes              |
| MeanPIM           | Mean server time of a single job at PIM                 | Yes              |
| NumberOfListeners | Number of listeners in a single node                    | Yes              |
| NumberOfThreads   | Number of threads in a single node                      | Yes              |
| QueueLengthAOC    | Maximum number of jobs in the queue of AOC              | Yes              |
| QueueLengthOR     | Maximum number of jobs in the queue of OR               | Yes              |
| QueueLengthPIM    | Maximum number of jobs in the queue of PIM              | Yes              |
| QueueLengthSystem | Maximum number of jobs in the queue to enter the system | Yes              |
| ResourceAOC       | Maximum number of jobs AOC can handle                   | Yes              |
| ResourceOR        | Maximum number of jobs OR can handle per node           | Yes              |
| ResourcePIM       | Maximum number of jobs PIM can handle                   | Yes              |
| TTLqueuesystem    | time to live at incoming queue                          | Yes              |

TABLE 4.4: Input variables

| Name                    | Meaning   |
|-------------------------|---|
| Percentage blocked jobs | Percentage of blocked jobs, based on scale between 0 and 1. |
| Time AOC2               | Average server time at AOC                                  |
| Time IOH1               | Average server time at IOH                                  |
| Time OR2                | Average server time OR                                      |
| Time PIM2               | Average server time PIM                                     |
| Time System 2           | Average total time spend in the system.                     |

TABLE 4.5: Record variables: Tally statistics.

results between two simulations with different lengths are almost identical, the shortest length is already enough. The same holds for the warm-up period. Try different settings for the warm-up period and just compare the results.

In the standard model we fix the parameters as given by ING Bank. The documentation and a test run gives us the results. We also simulate some other scenarios which we will compare with the standard model. We will compare the results to determine the bottleneck. We will start with a look at the standard model. Figure 4.14 shows us the number of incoming jobs compared to the number of outgoing jobs without drop-offs. With the lower load, there are almost no drop-offs. The number of drop-offs increase from a load of 55 requests per second. From then, the number of outgoing jobs will stagnate. So there is a bottleneck in the system which can only handle at most 55 requests per second.

| Name                      | Meaning   |
|---------------------------|---|
| Name                      | Meaning   |
| Drop Off AOC              | Total number of drop offs because the queue at AOC is full.                           |
| Drop Off AOC queue TTL    | Total number of drop offs at the queue of AOC because of the time to live is expired. |
| Drop Off AOC TTL          | Total number of drop offs at AOC because of the time to live is expired.              |
| Drop Off IOH              | Total number of drop offs at IOH because of the time to live is expired.              |
| Drop Off OR               | Total number of drop offs because the queue at OR is full.                            |
| Drop Off OR TTL           | Total number of drop offs at OR because of the time to live is expired.               |
| Drop Off PIM              | Total number of drop offs because the queue at PIM is full.                           |
| Drop Off PIM queue TTL    | Total number of drop offs at the queue of PIM because of the time to live is expired. |
| Drop Off PIM TTL          | Total number of drop offs at PIM because of the time to live is expired.              |
| Drop Off system           | Total number of drop offs because the incoming queue is full.                         |
| Drop off TTL queue system | Total number of drop offs because the time to live of the incoming queue is expired.  |
| Incoming request          | Total number of incoming requests   |
| Outgoing request          | Total number of outgoing requests.  |
| Request AOC               | Total number of request at AOC  |
| Request node 1            | Total number of request in node 1   |
| Request node 2            | Total number of request in node 2   |
| Request node 3            | Total number of request in node 3   |
| Request node 4            | Total number of request in node 4   |
| Request OR                | Total number of request at OR   |
| Request PIM               | Total number of requests at PIM   |

TABLE 4.6: Record variables: Counters

We will therefore analyze the drop-offs moments a bit closer. It turns out that almost all time-outs are because the Time to live at the incoming queue or at PIM are exceeded. This can give us multiple options for the bottleneck. Therefore we will analyse the drop-offs at these two points more closely

Figure 4.15 shows us the results. It looks like PIM can not handle the jobs any more. Therefore the queue or PIM grows, and jobs leave PIM since the Time to live is exceeded. Since all jobs in the queue of PIM eat up the sessions, at some moment, there are no free sessions and therefore jobs have to wait in the incoming queue for a larger period and thus at some stage, most jobs wait very long and these are kicked out since the time to live on this queue exceeded. This suggests that PIM is causing the bottleneck.

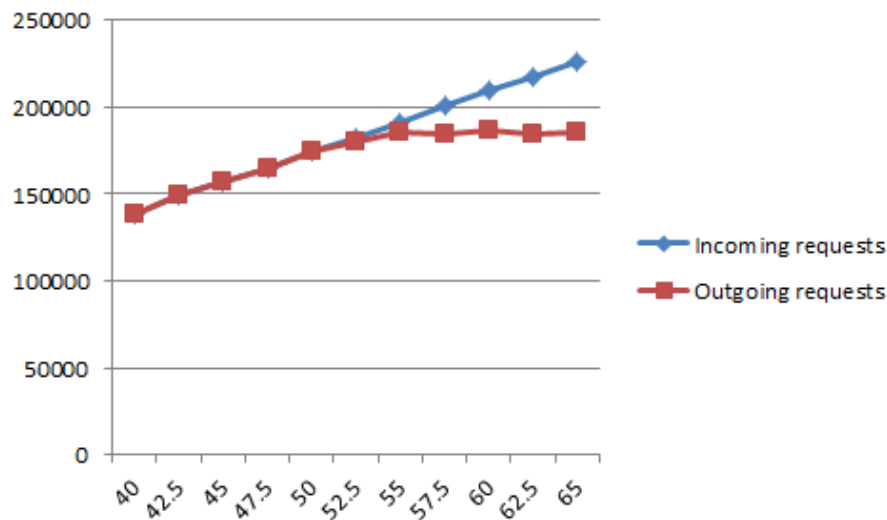


FIGURE 4.14: Blocking under normal scenario

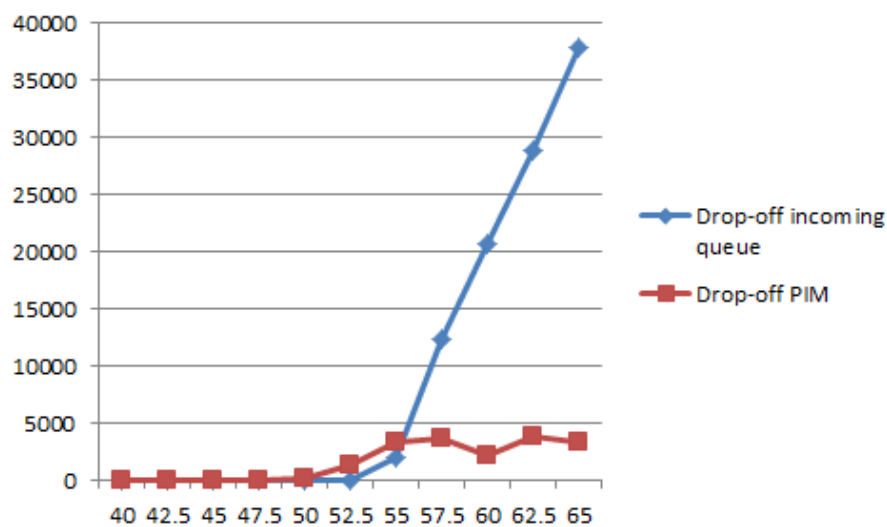


FIGURE 4.15: Blocking at incoming queue compared to blocking at PIM

We have decided to define 5 different scenario and we will compare the results of these scenarios with the standard scenario.

1. Faster AOC - We decrease the mean server time at AOC with 10%.
2. Faster IOH - We decrease the mean server time at IOH with 10%.
3. Faster OR - We decrease the mean server time at OR with 10%.
4. Faster PIM - We decrease the mean server time at PIM with 10%.
5. Extra sessions - We use twice the number of sessions.



These scenarios lead us to some notable results which are shown in figure 4.16. Four of

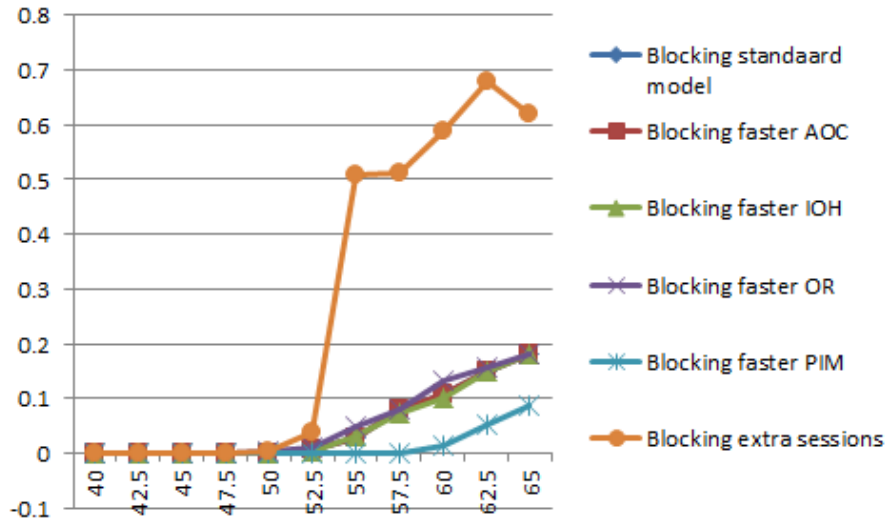


FIGURE 4.16: Blocking under the standard model and 5 different scenarios

the six scenarios behave roughly speaking the same. There is not that much difference. A faster PIM will increase the performance of the system. This means that PIM seems to be the bottleneck of this process. But it is quite remarkable that increasing the number of sessions will decrease the performance of the system enormously. This can be explained as follows. Since the number of sessions is increased, the number of jobs in the system can be larger. Since PIM cannot handle all the work any more, the queue of PIM grows. More jobs in the queue mean a higher waiting time. And therefore a much higher probability that the time to live at the queue is exceeded. It is much better for the jobs to wait in the incoming queue, then in the queue of PIM. Therefore the number of drop-offs will increase.



## Chapter 5

# Markov chains

In the previous chapter we have modeled a simulation for the CreateOrder service. You can model a queueing network in much detail with a simulation. But there is one problem: It takes a long time to get an answer and the answer contains always a bit of uncertainty. A longer run or more replications reduce the uncertainty, but is even more time consuming. Another approach to model a queueing network is to use Markov chains. This approach is much faster, but we have to simplify our model a bit. In this chapter we will develop the mathematical framework of the Markov chains.

We will start the development with the Markov chains, a specific stochastic process defined on a finite state space and in discrete time. Later on we will also allow an infinite state space and a continuous time scale. At the end we will see why this theory is useful in queueing networks and we will apply the theory on our model for the CreateOrder service.

### 5.1 Markov chain

A Markov chain [5, 6] is some specific stochastic process. The next step of the process does not depend on the history of the process. The probability distribution only depends on the current state.

**Definition 5.1** (Markov Chain). A stochastic process  $(X_t, t = 0, 1, 2, \dots)$  on a finite state space  $\Omega$  is called a Markov Chain if the Markov property holds i.e.

$$P(X_{t+1} = x_{t+1} \mid X_0 = x_0, X_1 = x_1, \dots, X_t = x_t) = P(X_{t+1} = x_{t+1} \mid X_t = x_t) \quad (5.1)$$

under the assumption that both conditional probabilities are well defined, i.e.

$$P(X_0 = x_0, X_1 = x_1, \dots, X_t = x_t) > 0 \quad (5.2)$$

**Definition 5.2** (Time-homogeneous Markov chains). A Markov chain is a Time-homogeneous Markov chains if the process does not depend on  $t$ , i.e.

$$P(X_{t+1} | X_t = x_t) = P(X_t | X_{t-1} = x_{t-1}) \quad (5.3)$$

for all  $t$ .

A Markov chain is fully determined if we have the following three aspects:

1. A state space  $\Omega$ . A state space is a set of possible outcomes of the process.
2. An initial distribution  $\pi_0$ . The initial distribution determines the starting point at  $t = 0$ .
3. A transition matrix  $P(t)$ . This matrix determines the probability laws. Element  $p_{ij}(t)$  determines the probability of jumping from state  $i$  to  $j$ . Matrix  $P(t)$  is stochastic, i.e.  $p_{ij}(t) \geq 0$  and  $\sum_{j \in \Omega} p_{ij}(t) = 1$  for all  $i \in \Omega$ .

In this thesis we will only look at time-homogeneous Markov chains.

Let us have a look at an example of a Markov chain. A common example of a Markov chain is the story of the frog and two ponds. At  $t = 0$  the frog starts at pond 1. At each moment ( $t = 1, 2, \dots$ ) the frog has each day the opportunity to jump to the other pond. If the frog is at pond 1, this probability is  $0 < p_1 < 1$ , if the frog is at pond 2, this probability will be  $0 < p_2 < 1$ . This process is a Markov chain with state space  $\Omega = \{1, 2\}$ ,  $\pi_0 = \{1, 0\}$  and  $P = \begin{Bmatrix} 1 - p_1 & p_1 \\ p_2 & 1 - p_2 \end{Bmatrix}$  given the frog starts at pond 1 at  $t = 0$ .

We are interested in the probability that the frog is at pond 1 at time  $t = 2$ . We can calculate this probability with simple probability theory. There are two possibilities: The frog jumped at  $t = 1$  to pond 2 and jumped back at  $t = 2$  or the frog has not jumped yet. This probability is  $(1 - p_1)^2 + p_1 p_2$ .

We could also use Markov Chains. Let us have a look at  $\pi_0 P = (1 - p_1, p_1)$ . This is equal to  $(P(X_1 = 1 | X_0 = 1), P(X_1 = 2 | X_0 = 1))$ . If we apply  $P$  again we get  $\pi_0 P^2 = (p_1 p_2 + (1 - p_1)^2, (1 - p_1) p_1 + p_1 (1 - p_2))$ . This is equal to  $(P(X_2 = 1 | X_0 = 1), P(X_2 = 2 | X_0 = 1))$ . We can continue and see  $\pi_0 P^n = (P(X_n = 1 | X_0 = 1), P(X_n = 2 | X_0 = 1))$ . This holds for each Markov chain. This leads us to the question what has happened what does happens in the long run, i.e. what is  $\lim_{n \rightarrow \infty} \pi_0 P^n$ ? Does the limit exist?

Does it depend on  $\pi_0$ ? Let us assume the limit exists, say  $\pi = \lim_{n \rightarrow \infty} \pi_0 P^n$ . Then it must satisfy  $\pi = \pi P$ . In our example of the frog it turns out that  $\pi = (\frac{p_2}{p_1+p_2}, \frac{p_1}{p_1+p_2})$ , which is independent of the initial distribution. In this example the limiting distribution exists and is unique, but this is not always the case. To answer this question, we had to solve a system of linear equations and a system of linear equations will not always give a unique solution. It could be even worse: Let us consider the same example, but now with  $p_1 = p_2 = 1$ . This means: The frog jumps every day to the other pond. The probability that the frog is on pond 1 is 0 if it is an odd day and 1 if it is an even day. The sequence 1, 0, 1, 0, 1, 0, 1, ... does not converge. So the limit does not even exist! Therefore we cannot assume that a Markov chain has a limiting distribution. Therefore we have to think about some properties of a Markov Chain under which we can prove that the limiting distribution exists. Therefore we need some definitions.

### 5.1.1 Properties

**Definition 5.3** (Stationary distribution). A distribution is called stationary if  $\pi = \pi P$ . An equivalent definition is given by  $\pi(y) = \sum_{x \in \Omega} \pi(x) P(x, y)$  for all  $x, y \in \Omega$ .

**Definition 5.4** (Accessible). A state  $y$  is accessible from state  $x$  in a Markov chain  $(X_t, t = 0, 1, 2, \dots)$  if there exists a  $n_{xy} \geq 0$  (these values can be different for each pair  $(x, y)$ ) s.t.

$$P(X_{n_{xy}} = y \mid X_0 = x) > 0. \quad (5.4)$$

**Definition 5.5** (Commute). State  $i$  and  $j$  commute if  $i$  is accessible from  $j$  and  $j$  is accessible from  $i$ .

**Definition 5.6** (Irreducible). A Markov chain  $(X_t, t = 0, 1, 2, \dots)$  is irreducible if each state is accessible from any other state, i.e. for any  $x, y \in \Omega$  there exist  $n_{xy} \geq 0$  (possibly depending on  $x$  and  $y$ ) s.t.

$$P(X_{n_{xy}} = y \mid X_0 = x) > 0. \quad (5.5)$$

**Definition 5.7** (Period). A state has period  $m$  if

$$m = \gcd\{n \geq 1 : P(X_n = i \mid X_0 = i) > 0\}. \quad (5.6)$$

**Definition 5.8** (Aperiodic Markov chain). A Markov chain is called aperiodic if all states have period 1.

**Lemma 5.9.** Let  $(X_t, t = 0, 1, 2, \dots)$  be a Markov chain. Let  $M(i) = \{n \geq 1 : P(X_n = i \mid X_0 = i) > 0\}$ . If  $P$  is irreducible, then  $\gcd(M(i)) = \gcd(M(j))$  for all  $i, j \in \Omega$ .

*Proof.* Fix some states  $i$  and  $j$ . Since  $X$  is irreducible there exist  $m$  and  $n$  such that  $p_{ij}^m > 0$  and  $p_{ji}^n > 0$ . Let  $k = m + n$ . Then  $k \in M(i) \cap M(j)$ . We also have  $M(i) + k \subset M(j)$ . This means that the  $\gcd(M(j))$  divides also all elements in  $M(i)$ . Therefore we know  $\gcd(M(j)) \leq \gcd(M(i))$ . We can use the same argument to prove  $\gcd(M(j)) \geq \gcd(M(i))$  and thus  $\gcd(M(j)) = \gcd(M(i))$   $\square$

It turns out that this will be an useful result. It will be the case that we are mainly looking at aperiodic Markov chains. Using this lemma, it is sufficient to only check the period of one single state, instead of all states. Let us consider our frog example again. The frog may not jump and will be in the same state as the previous day. This is enough to determine that this state is aperiodic and thus the Markov chain will be aperiodic.

**Definition 5.10** (First return time). The first return time of a random variable is given by

$$T_i = \inf\{n \geq 1 : X_n = i \mid X_0 = i\}. \quad (5.7)$$

**Definition 5.11** (Recurrent). A state  $i$  is recurrent if the probability of returning to state  $i$  is equal to one. A state is recurrent if  $P(T_i < \infty) = 1$ .

**Definition 5.12** (Transient). A state  $i$  is transient if there is a non-zero probability of never returning back, i.e.  $P(T_i < \infty) < 1$

**Definition 5.13** (Positive recurrent). A state  $i$  is positive recurrent if the mean recurrence time is finite, that is  $E[T_i] < \infty$ .

### 5.1.2 The limiting distribution

**Lemma 5.14.** For any states  $x$  and  $y$  of an irreducible Markov chain, we have  $E(T_y \mid X_0 = x) < \infty$ .

*Proof.* From the definition of an irreducible Markov chain there exist an integer  $r > 0$  and a real  $\epsilon > 0$  such that for any  $v, w \in \Omega$  there exists a  $j \leq r$  with  $P^j(v, w) > \epsilon$ . Thus for any value of  $X_t$ , the probability of hitting state  $y$  at a time between  $t$  and  $t + r$  is at least  $\epsilon$ . For  $k > 0$  we have

$$P(T_y > kr \mid X_0 = x) = P(T_y > (k-1)r \text{ and } T_y \notin [(k-1)r, kr] \mid X_0 = x) \quad (5.8)$$

$$\leq (1 - \epsilon)P(T_y > (k-1)r \mid X_0 = x). \quad (5.9)$$

Repeating this argument yields

$$P(T_y > kr \mid X_0 = x) \leq (1 - \epsilon)^k. \quad (5.10)$$

Since  $P(T_y > t \mid X_0 = x)$  is a decreasing function of  $t$ , we have

$$E(T_y \mid X_0 = x) = \sum_{t \geq 0} P(T_y > t \mid X_0 = x) \quad (5.11)$$

$$\leq \sum_{t \geq 0} r P(T_y > kr \mid X_0 = x) \quad (5.12)$$

$$\leq r \sum_{t \geq 0} (1 - \epsilon)^k < \infty \quad (5.13)$$

□

**Theorem 5.15.** *Let  $P$  be a transition matrix of an irreducible Markov chain. Then there exists a probability distribution  $\pi$  on  $\Omega$  such that  $\pi = \pi P$ . Furthermore  $\pi(x) > 0$  for all  $x \in \Omega$  and  $\pi(x)$  is given by  $\pi(x) = \frac{1}{E(T(x) \mid X_0 = x)}$ .*

*Proof.* Let  $v$  be some arbitrary state of a Markov chain. We will define a variable  $\tilde{\pi}(y)$  that determines the time a chain spends in state  $y$  between two visits at state  $v$ . In mathematical formulas we will get

$$\tilde{\pi}(y) = E(\text{number of visits of } y \text{ before returning to } v) \quad (5.14)$$

$$= \sum_{t \geq 0} P(X_t = y, T_v > t \mid X_0 = z). \quad (5.15)$$

Our previous lemma ensures that  $\tilde{\pi}(y) \leq E(T_z \mid X_0 = z) < \infty$  for all  $y \in \Omega$ . We will check the definition of a stationary distribution:

$$\tilde{\pi}(y) = \sum_{x \in \Omega} \tilde{\pi}(x) P(x, y) \quad (5.16)$$

$$= \sum_{x \in \Omega} \sum_{t \geq 0} P(X_t = x, T_v > t \mid X_0 = z) P(x, y) \quad (5.17)$$

$$= \sum_{t \geq 0} P(X_{t+1} = y, T_v > t + 1) \quad (5.18)$$

$$= \sum_{t \geq 1} P(X_t = y, T_v > t) \quad (5.19)$$

where the last steps follows from changing the order of summation. This expression is already almost equal to  $\tilde{\pi}(y)$ . In fact,

$$\sum_{t \geq 1} P(X_t = y, T_v > t) = \tilde{\pi}(y) - P(X_0 = y, T_v > 0 \mid X_0 = v) \quad (5.20)$$

$$+ \sum_{t \geq 1} P(X_t = y, T_v = t \mid X_0 = v) \quad (5.21)$$

$$= \tilde{\pi}(y) - P(X_0 = y \mid X_0 = v) + P(X_{T_v} = y) \quad (5.22)$$

$$= \tilde{\pi}(y) \quad (5.23)$$

where the last steps follows from the following observation. Let us have a look at the case  $y = v$ . The last two terms will be equal to one. If  $y \neq v$ , both terms are equal to zero. Therefore the last expression holds and we have shown that  $\tilde{\pi} = \tilde{\pi}P$ .

$\tilde{\pi}$  may not be a probability distribution. Therefore we have to normalize by  $\sum_{x \in \Omega} \tilde{\pi}(x) = E(T(v) | X_0 = v)$  and thus  $\pi(x) = \frac{\tilde{\pi}(x)}{E(T(v) | X_0 = v)}$  satisfies  $\pi = \pi P$ . Consider the case  $x = v$ , then it follow that  $\pi(x) = \frac{1}{E(T(x) | X_0 = x)}$ . Since  $v$  was arbitrary, this holds for every  $x$ .  $\square$

This is our first useful result. An irreducible Markov chain has a stationary distribution. At this moment we do not know if we have an unique solution and whether the limiting distribution converges to this stationary distribution. It turns out that this is not always the case. Let us have again a look at our example of the frog. We will again have a look at the case the frog jumps every day to the other pond. This example is an irreducible Markov chain and thus we can apply the above theory. We find  $\pi = (\frac{1}{2}, \frac{1}{2})$  as stationary distribution. It is indeed true that the frog is in half of the days in the first pond and the other half of the days in the second pond. But this is not equal to  $\lim_{t \rightarrow \infty} \pi_0 P^t$ . So the existence of a stationary distribution is not enough to conclude that the process will converges to the stationary distribution of the theory.

In order to continue we will have to introduce a metric for measuring the distance between two distributions. This will be the total variation distance.

**Definition 5.16** (Total variance distance). The total variance distance between two probability distributions  $\mu$  and  $\nu$  on the state space  $\Omega$  is given by

$$\|\mu - \nu\|_{TV} = \max_{A \subset \Omega} |\mu(A) - \nu(A)|. \quad (5.24)$$

**Lemma 5.17.** Let  $\mu$  and  $\nu$  be two probability distributions on  $\Omega$ . Then

$$\|\mu - \nu\|_{TV} = \frac{1}{2} \sum_{x \in \Omega} |\mu(x) - \nu(x)|. \quad (5.25)$$

*Proof.* Let  $B = \{x : \mu(x) \geq \nu(x)\}$  and  $A \supset \Omega$  some event. Then we must have

$$\mu(A) - \nu(A) = \mu(A \cap B) + \mu(A \cap B^c) - (\nu(A \cap B) + \nu(A \cap B^c)) \quad (5.26)$$

$$= \mu(A \cap B) - \nu(A \cap B) + \mu(A \cap B^c) - \nu(A \cap B^c) \quad (5.27)$$

$$\leq \mu(A \cap B) - \nu(A \cap B) + 0 \quad (5.28)$$

$$\leq \mu(B) - \nu(B). \quad (5.29)$$

The last expression is true since  $\mu(A^c \cap B) - \nu(A^c \cap B) > 0$  by definition of the set  $B$ . With the same reasoning it can be shown that  $\nu(A) - \mu(A) \leq \mu(B^c) - \nu(B^c)$ . Both upper



limits are in fact the same upper limits. This follows from the fact that the probability measures add up to one. The upper limit will be obtained if  $A = B$  or  $A = B^c$ . Thus

$$\|\mu - \nu\|_{TV} = \frac{1}{2}(\mu(B) - \nu(B) + \nu(A) - \mu(A)) \quad (5.30)$$

$$= \frac{1}{2} \sum_{x \in \Omega} |\mu(x) - \nu(x)|. \quad (5.31)$$

□

**Theorem 5.18.** *Let  $P$  be the transition matrix of an irreducible and aperiodic Markov chain and let  $\pi$  be a stationary distribution. Then there exist constants  $a \in (0, 1)$  and  $C > 0$  such that*

$$\max_{x \in \Omega} \|P^t(x, \cdot) - \pi\|_{TV} \leq Ca^t. \quad (5.32)$$

*Proof.* Since  $P$  is irreducible and aperiodic there exists  $r$  such that  $P^r$  contains only strictly positive entries. Let  $\Pi$  be a  $|\Omega| \times |\Omega|$  matrix with each row containing the row vector  $\pi$ . Fix  $\delta > 0$  such that  $P^r(x, y) \geq \delta\pi(y)$  for all  $x, y \in \Omega$ . Let  $\theta = 1 - \delta$ . Let  $Q$  be a stochastic matrix such that  $P^r = (1 - \theta)\Pi + \theta Q$ . We will prove by induction that  $P^{rk} = (1 - \theta^k)\Pi + \theta^k Q^k$  for  $k \geq 1$ . It holds for  $k = 1$  by assumption. Let us assume it holds for  $k = n$ . We will prove that it also holds for  $k = n + 1$ . We start with

$$P^{r(n+1)} = P^{rn} P^r \quad (5.33)$$

$$= ((1 - \theta^n)\Pi + \theta^n Q^n) P^r \quad (5.34)$$

$$= ((1 - \theta^n)\Pi + \theta^n Q^n)((1 - \theta)\Pi + \theta Q). \quad (5.35)$$

Expanding the brackets and using  $\Pi P^r = \Pi$  and  $Q^n \Pi = \pi$  yields

$$P^{r(n+1)} = P^{rn} P^r = (1 - \theta^n)\Pi P^r + (1 - \theta)\theta^n Q^n \Pi + \theta^{n+1} Q^{n+1} \quad (5.36)$$

$$= (1 - \theta^{n+1})\Pi + \theta^{n+1} Q^{n+1} \quad (5.37)$$

which is the expression that we were looking for.

We will rearrange the terms and multiply all terms with  $P^j$ . Then we get

$$P^{rk+j} - \Pi = \theta(Q^k P^j - \Pi) \quad (5.38)$$

for some  $k$ . We can see this equation as two matrices who must be equal. We will sum over the elements of some row  $z$  on both sides and divide by two, so the expression on the left side is the total variance distance, the right side must be smaller than the largest possible total variation distance between the two distributions, which is 1, multiplied by

the constant  $\theta^k$ . This is

$$\max_{x \in \Omega} \|P^t(z, \cdot) - \pi\|_{TV} \leq \theta^k, \quad (5.39)$$

which completes the proof.  $\square$

These results give us that a Markov chain converge to a unique limiting distribution if the Markov chain is aperiodic and irreducible. If we can find a stationary distribution for such Markov chain, we immediately know that this stationary distribution is unique and the limiting distribution will converge to this distribution. This means we can easily calculate the limiting distribution and other related statistics based on the limiting distribution. We already have seen a way to calculate the limiting distribution: Show that a Markov chain is aperiodic and irreducible and solve a system of linear equations. On the other hand, we know there is one unique solution which leads to a probability distribution.

These results can be extended to allow not only finite state spaces, but also countable state spaces. For state spaces which are countable being an aperiodic and irreducible Markov chain is not enough. The Markov chain must also be positive recurrent. We will not discuss Markov chains with a countable large state space. We are more interested in stochastic processes with a continuous time scale. Therefore our next step will be adding a continuous time scale to the Markov process

### 5.1.3 Time reversible

Let us have a look at a Markov chain for which the stationary limiting distribution exists, and which is in operation for a long time, so that we can assume the process is stationary. Let us trace back such a Markov chain backward in time. This means, we start at a certain  $n$  and look at the sequence  $X_n, X_{n-1}, X_{n-2}, \dots$ . This chain will be called the reversed process.

**Lemma 5.19.** *Let  $\{X_n; n = 0, 1, 2, \dots\}$  be a stationary Markov chain with limiting distribution  $\pi$  and transition matrix  $P$ . The reversed process of such a Markov Chain is again a Markov chain with transition matrix  $R = (r_{ij})$  with  $r_{ij} = \frac{\pi_j p_{ij}}{\pi_i}$ .*

*Proof.* We will start by proving that the transition matrix is correct.

$$r_{ij} = P(X_n = j \mid X_{n-1} = i) \quad (5.40)$$

$$= \frac{P(X_n = j, X_{n-1} = i)}{P(X_{n-1} = i)} \quad (5.41)$$

$$= \frac{P(X_n = j)P(X_{n-1} = i \mid X_n = j)}{P(X_{n-1} = i)} \quad (5.42)$$

$$= \frac{\pi_j p_{ij}}{\pi_i} \quad (5.43)$$

The other step is to prove that the reversed chain has the Markov property, i.e.

$$P(X_n = x_n \mid X_{n+1} = x_{n+1}, X_{n+2} = x_{n+2}, \dots) = P(X_n = x_n \mid X_{n+1} = x_{n+1}). \quad (5.44)$$

Since  $X_0, X_1, \dots$  is a Markov chain it follows from the definition of the conditional distribution that the future  $X_{n+2}, X_{n+3}, \dots$  given the present state  $X_{n+1}$  is independent of the past state  $X_n$ . Independence is a symmetric relationship. This means: If A is independent of B, then B is independent of A. Therefore the conditional distribution of the past state  $X_n$  given the current state  $X_{n+1}$  is independent of the future  $X_{n+2}, X_{n+3}, \dots$ . This is exactly what we are looking for. Therefore the reversed chain has the Markov property.  $\square$

**Definition 5.20** (Time reversible Markov chain). Let  $\{X_n; n = 0, 1, 2, \dots\}$  be a stationary Markov chain with limiting distribution  $\pi$  and transition matrix  $P$ . Let  $\{Y_n; n = 0, 1, 2, \dots\}$  be the reversed process of Markov chain  $X$  with transition matrix  $R$ . If  $p_{ij} = r_{ji}$  for all  $i, j$ , then the Markov chain  $X$  is called time reversible.

From the definition of  $r_{ij} = \frac{\pi_j p_{ij}}{\pi_i}$  it follows that a Markov chain is time reversible if  $\pi_i p_{ij} = \pi_j p_{ji}$  for all  $i, j$ .

**Lemma 5.21.** Let  $P$  be a transition matrix. Let  $x_i$  be a set of non-negative numbers s.t.

$$\sum_i x_i = 1 \quad (5.45)$$

$$x_i p_{ij} = x_j p_{ji}, \quad (5.46)$$

then the Markov chain corresponding to the transition matrix  $P$  is time reversible and the limiting distribution is given by  $\pi_i = x_i$ .

*Proof.* Summing  $x_i p_{ij} = x_j p_{ji}$  over all  $i$  gives

$$\sum_i x_i p_{ij} = \sum_i x_j p_{ji} \quad (5.47)$$

$$= x_j \sum_i p_{ji} \quad (5.48)$$

$$= x_j. \quad (5.49)$$

Since the limiting distribution is the unique solution of this system of linear equations and the normalization equation, we have  $x_i = \pi_i$  and thus we have  $\pi_i P_{ij} = \pi_j P_{ji}$  and thus is the Markov chain time reversibly.  $\square$

## 5.2 Continuous-time Markov chain

Now we know we can solve a Markov chain, we will change the time scale to continuous. This process is called a continuous-time Markov chain (CTMC). A Continuous-time Markov chain is almost the same process as a discrete Markov chain. The main difference is the time scale. A Markov chain uses a discrete time scale. A Continuous-time Markov chain [6, 7] uses a continuous time scale. We will also analyze this system and we will see that the CTMC will fit in for some queueing models.

**Definition 5.22** (Continuous-time Markov chain). A stochastic process  $(X_t, t \geq 0)$  on a finite state space  $\Omega$  is called a Continuous-time Markov chain if for  $0 = t_0 < t_1 < \dots < t_{n+1}$  and  $x_0, x_1, \dots, x_{n+1} \in \Omega$  it holds

$$P(X_{t_{n+1}} = x_{t_{n+1}} \mid X_0 = x_0, X_{t_1} = x_1, \dots, X_{t_n} = x_n) = P(X_{t_{n+1}} = x_{t_{n+1}} \mid X_t = x_t) \quad (5.50)$$

under the assumption that both conditional probabilities are well defined, i.e.

$$P(X_0 = x_0, X_{t_1} = x_1, \dots, X_{t_n} = x_n) > 0 \quad (5.51)$$

It might be a surprise, but this process is related to the exponential distribution. The time that a process is in some state is exponentially distributed. This is the only distribution for which the definition of the continuous-time Markov chain holds. This is related with the memoryless property of the exponential distribution. All other continuous time distributions do not have this property.

Analogue to the Markov Chain, we will also only look at time homogeneous CTMCs. This means that the behavior over time does not depend on the time itself.

Analogue to the Markov chain, we will also be looking for the limiting distribution of a CTMC. We will mainly looking at methods to calculate the limiting distribution. We

will assume that the limiting distribution exist. This assumption will hold under certain conditions that are similar to the conditions for the discrete time Markov chain. It turns out that the limiting distribution exists if all states communicate with each other and the CTMC must be positive recurrent. So in the next part of the section we will mainly look for an expression to calculate the limiting distribution.

The CTMC can be described in different ways. We gave above one description, but we could also describe the process as a Markov Jump Process. Let  $\Omega$  be a discrete state space. The process jumps from state  $i$  to state  $j$  with probability  $p_{ij}$ . Note that we assume  $p_{ii} = 0$ . The time before the process make its next job is exponentially distributed with parameter  $v_i$ . We will also speak of the rate  $v_i$ . So the mean time a job spends in state  $i$  is  $\frac{1}{v_i}$ . This process fits in our definition of a CTMC and every CTMC can be described this way.

For describing a Markov chain we had to define an initial distribution, a transition matrix  $P$  and a state space. For a CTMC we still have to describe the state space and the initial distribution. But we have to refine our transition matrix  $P$ . It is not enough to only describe in which state the system is in, we must also define how long a job spends on average in a state.

Therefore we have to define  $q_{ij} = v_i p_{ij}$  for  $j \neq i$  and  $q_{ii} = q_i = -\sum_{j \neq i} q_{ij}$ . Since  $v_i$  describes the rate at which a job leaves the state and  $p_{ij}$  describes the next state,  $q_{ij}$  can be seen as the rate at which the process leaves state  $i$  and goes to state  $j$ . Therefore they are also called the instantaneous transition rates. From this definition it follows that  $v_i = \sum_{j \neq i} v_i p_{ij} = \sum_{j \neq i} q_{ij}$  and  $p_{ij} = \frac{q_{ij}}{v_i} = \frac{q_{ij}}{\sum_{j \neq i} q_{ij}}$ . This calculation shows that  $q_{ij}$  contains all information of  $v$  and  $p$ .

It turns out that Matrix  $Q = (q_{ij})$  is the transition matrix for a CTMC. So if we want to describe a CTMC it is enough to give this transition matrix  $Q$ , the initial distribution.

**Definition 5.23** (Transition Matrix). A  $n \times n$  matrix  $Q = (q_{ij})$  is a transition matrix if

1.  $0 \leq -q_i \leq \infty$ ,
2.  $0 \leq q_{ij}$  for all  $i \neq j$ ,
3. and  $\sum_j q_{ij} = 0$ .

### 5.2.1 The limiting distribution

**Lemma 5.24.**     •  $\lim_{h \rightarrow 0} \frac{1 - p_{ii}(h)}{h} = -q_i$ .

- $\lim_{h \rightarrow 0} \frac{p_{ij}}{h} = q_{ij}$  if  $i \neq j$ .

*Proof.* Let us start with the first expression. The amount of time until the next transition is exponentially distributed. It follows that the probability of more than two transitions in time  $h$  is  $o(h)$ . This gives us that  $1 - p_{ii}(h)$  is the probability that the process was in state  $i$  at time 0, but is not any more in state  $i$  at time  $h$ . For same values of  $h$  this probability is equal to  $-q_i h$ . Therefore  $1 - p_{ii}(h) = -q_i h + o(h)$ . this proves the first part.

Analogously, we can say  $p_{ij}(h)$  is the probability that a job jumps before time  $h$  from state  $i$  to state  $j$ . The probability of jumping in this interval is equal to  $h v_i$  and the probability of jumping to state  $j$  is  $p_{ij}$ . The probability of multiple jumps is again  $o(h)$ . Therefore  $p_{ij}(h) = h v_i p_{ij} + o(h)$ .  $\square$

**Lemma 5.25.** *The Chapman-Kolmogorov equations are given by*

$$p_{ij}(t + s) = \sum_{k \in \Omega} p_{ik}(t) p_{kj}(s) \quad (5.52)$$

for all  $s \geq 0$  and  $t \geq 0$ .

*Proof.*

$$p_{ij}(t + s) \quad (5.53)$$

$$= P(X(t + s) = j \mid X(0) = i) \quad (5.54)$$

$$= \sum_{k \in \Omega} P(X(t + s) = j \mid X(t) = k, X(0) = i) P(X(t) = k \mid X(0) = i) \quad (5.55)$$

$$= \sum_{k \in \Omega} P(x(t + s) = j \mid X(t) = k, X(0) = i) P(X(t) = k \mid X(0) = i) \quad (5.56)$$

$$= \sum_{k \in \Omega} P(x(t + s) = j \mid X(t) = k) P(X(t) = k \mid X(0) = i) \quad (5.57)$$

$$= \sum_{k \in \Omega} P(x(s) = j \mid X(0) = k) P(X(t) = k \mid X(0) = i) \quad (5.58)$$

$$= \sum_{k \in \Omega} p_{ik}(t) p_{kj}(s) \quad (5.59)$$

$\square$

**Theorem 5.26.** *The Kolmogorov's backward equations are given by*

$$p'_{ij}(t) = \sum_{k \neq i} q_{ik} P_{kj}(t) + q_i P_{ij}(t) \quad (5.60)$$

and the Kolmogorov's forward equations are given by

$$p'_{ij}(t) = \sum_{k \neq i} q_{kj} P_{ik}(t) + q_j P_{ij}(t) \quad (5.61)$$

for all states  $i, j \in \Omega$  and  $t > 0$ .

*Proof.* We use the Chapman-Kolmogorov equations to derive

$$p_{ij}(h+t) - p_{ij}(t) = \sum_{k \in \Omega} p_{ik}(t)p_{kj}(h) - p_{ij}(t) \quad (5.62)$$

$$= \sum_{k \in \Omega, k \neq i} p_{ik}(t)p_{kj}(h) - (1 - p_{ii}(h))p_{ij}(t) \quad (5.63)$$

and thus

$$\lim_{h \rightarrow 0} \frac{p_{ij}(t+h) - p_{ij}(t)}{h} = \lim_{h \rightarrow 0} \sum_{k \in \Omega, k \neq i} p_{ik}(t) \frac{p_{kj}(h)}{h} - \frac{1 - p_{ii}(h)}{h} p_{ij}(t). \quad (5.64)$$

We may interchange the summation and the limit and we use Lemma 5.25 to obtain

$$p'_{ij}(t) = \sum_{k \neq i} q_{ik}p_{kj}(t) + q_i p_{ij}(t). \quad (5.65)$$

The proof of the Kolmogorov forward equations is similar. Note that we are looking at CTMC with a finite state space. These results can be extended to a countable state space.  $\square$

With the Kolmogorov forward and backward equations we have some tools to calculate the limiting distribution under the assumption that the limiting distribution exist. Define

$$p_j = \lim_{t \rightarrow \infty} p_{ij}(t) \quad (5.66)$$

and let us have a look at what happens if we take the limit of the Kolmogorov's forward equation, i.e.

$$\lim_{t \rightarrow \infty} p'_{ij}(t) = \lim_{t \rightarrow \infty} \sum_{k \neq i} q_{kj}P_{ik}(t) + q_j P_{ii}(t) \quad (5.67)$$

and interchange the summation and the limit. We obtain

$$\lim_{t \rightarrow \infty} p'_{ij}(t) = \sum_{k \neq j} q_{kj}P_k + q_j p_j. \quad (5.68)$$

Since  $p_{ij}(t)$  is bounded between 0 and 1 and converges to some value (since we assume that the limiting distribution exists), we know  $\lim_{t \rightarrow \infty} p'_{ij}(t) = 0$ . From here it follows

that

$$0 = \sum_{k \neq j} q_{kj} p_k + q_j p_j \text{ or} \quad (5.69)$$

$$-q_j p_j = \sum_{k \neq j} q_{kj} p_k \text{ for all states } j. \quad (5.70)$$

This system of equation combined with  $\sum_i p_i = 1$  can be used to solve the limiting distribution of a CTMC. These equations are called the balance equations. They can be interpreted as follows:  $-q_j p_j$  is the rate at which that process leaves state  $j$  and  $\sum_{k \neq j} q_{kj} p_k$  is the rate at which the process enters state  $j$ . These two rates are in balance in the limiting distribution.

### 5.2.2 Embedded Markov Chain

Trying to solve the balance equations is not the only way to solve the limiting distribution of a CTMC. Let us ignore the time and just consider the sequence of states in which the CTMC is in. This sequence is a Markov chain with transition probabilities  $p_{ij}$ . This Markov chain is called the embedded Markov chain. Let be  $\pi$  the limiting distribution of the embedded Markov chain.

We can use this solution to solve the limiting distribution of a CTMC. The limiting distribution  $\pi$  can be viewed as the proportion of transition of the process to state  $i$ . The time spent in state  $i$  is exponentially distributed with parameter  $-q_i$ . Therefore the mean time spent in state  $i$  during a visit is  $\frac{1}{-q_i}$ . The limiting distribution  $p_i$  is therefore proportional to the weighted averages of  $\pi$  with  $\frac{1}{-q_i}$  as weights. Intuitively speaking, this implies

$$p_i = \frac{\frac{\pi_i}{-q_i}}{\sum_j \frac{\pi_j}{-q_j}}. \quad (5.71)$$

If this is indeed the limiting distribution, it must satisfy

$$-q_j p_j = \sum_{k \neq j} q_{kj} p_k. \quad (5.72)$$

It turn out that this is true if

$$\pi_i = \sum_j \pi_j p_{ji} \text{ for all } i. \quad (5.73)$$

This is true since  $\pi$  is the limiting distribution of the embedded Markov Chain.



### 5.2.3 Time reversal

Embedded Markov chains are not only useful for determining the limiting distribution of a CTMC. It makes it also possible to extend the definition of a time reversal Markov chain to a time reversal CTMC. Let us first have a look at the reversed process of a CTMC. Recall: A Markov chain is time reversal if  $\pi_i p_{ij} = \pi_j p_{ji}$  holds for every  $i, j$ .

Let  $\{X_t, t \geq 0\}$  be a Markov chain operating long enough to be in steady state. Suppose we look back in time from some large value of  $T$ . We want to determine the probability structure of this reversed process. Define  $Y_t = X_{T-t}$  as the reversed process. Then

$$P(Y_{t+x} = i \text{ for all } x \in [0, s] \mid Y_t = i) \quad (5.74)$$

$$= P(X_{T-t-x} = i \text{ for all } x \in [0, s] \mid X_{T-t} = i) \quad (5.75)$$

$$= \frac{P(X_{T-t-x} = i \text{ for all } x \in [0, s])}{P(X_{T-t} = i)} \quad (5.76)$$

$$= \frac{P(X_{T-t-x} = i)e^{q_i s}}{P(X_{T-t} = i)} \quad (5.77)$$

$$= e^{q_i s} \quad (5.78)$$

where we used in the third step that the time spent in a state  $i$  is exponentially distributed with parameter  $-q_i$ . In the last step we used  $P(X_{T-t-x} = i) = P(X_{T-t} = i) = p_i$  for large  $T$ , because we are assuming that the process is in steady state. In other words, if we look backward in time for the amount of time a process spends in state  $i$ , it is also exponentially distributed with parameter  $-q_i$ . The reversed process is therefore also a CTMC!

**Definition 5.27** (Time reversal CTMC). A CTMC is a time reversal CTMC if the corresponding embedded discrete MC is a time reversible Markov chain.

**Lemma 5.28.** A CTMC is time reversal if  $p_i q_{ij} = p_j q_{ji}$  for all  $i, j$ .

*Proof.* The embedded Markov chain must be time reversible and thus  $\pi_i p_{ij} = \pi_j p_{ji}$  holds and  $p_i = \frac{\pi_i}{\sum_j \frac{-q_j \pi_j}{v_j}}$  is the limiting distribution of the CTMC and thus  $\pi_i = p_i v_i \sum_j \frac{\pi_j}{v_j}$ . Combining this expression with  $q_{ij} = v_i p_{ij}$  gives us  $p_i q_{ij} = p_j q_{ji}$  for all  $i, j$ .  $\square$

At this stage it is not clear why we are looking at time reversible CTMCs. If we make the step to queueing theory it turns out that time reversible CTMCs are very useful to analyze complex queueing networks.

## 5.3 Queueing theory

### 5.3.1 M/M/1

Some queueing networks [1, 8] can be described by a CTMC. Therefore we can use the results of the previous section for some queueing networks. An obvious queueing network which can be described as CTMC is the M/M/1 queue. customers arrives at rate  $\lambda$  and are served at rate  $\mu$ . Let us define the state space  $\Omega = \{0, 1, 2, \dots\}$  and let the transition matrix  $Q$  be defined as

$$Q = \begin{pmatrix} -\lambda & \lambda & & & & \\ \mu & -(\lambda + \mu) & \lambda & & & \\ & \mu & -(\lambda + \mu) & \lambda & & \\ & & \mu & -(\lambda + \mu) & \lambda & \\ & & & \mu & -(\lambda + \mu) & \lambda \\ & & & & & \ddots \end{pmatrix}. \quad (5.79)$$

This CTMC describes the number of customers in the M/M/1 queue. We can easily calculate the limiting distribution of the CTMC by using the balance equations. These give us the following system of equations:

$$\lambda p_0 = \mu p_1 \quad (5.80)$$

$$(\lambda + \mu)p_n = \mu p_{n+1} + \lambda p_{n-1} \text{ for } n \geq 1 \quad (5.81)$$

$$\sum_{n=0}^{\infty} p_n = 1 \text{ (normalization equation)}. \quad (5.82)$$

We can solve this system recursively. Define  $\rho = \frac{\lambda}{\mu}$ . We will assume that  $0 < \rho < 1$ , since it turns out that this is a condition for the existence of the limiting distribution. We can interpret this condition as follows:  $\mu$  is the rate at which the server processes the jobs and  $\lambda$  is the rate at which the customers enter the queue. This means that the mean server time must be smaller than the mean time between two arrivals. This seems a viable condition. Assume that this condition is violated. This means that the average number of jobs that enter the system is higher than the average number of jobs the server can handle. The queue will explode and thus the system is unstable and has no limiting distribution.

From the first equation we obtain  $p_1 = \rho p_0$ . Use this in the second equation, we get  $p_2 = \rho^2 p_0$ . We can continue and obtain  $p_n = \rho^n p_0$ . Using the normalization equation we found  $p_0 = \frac{1}{\sum_{n=0}^{\infty} \rho^n} = (1 - \rho)$  and thus  $p_n = (1 - \rho)\rho^n$  for  $n \geq 0$ . Given this limiting behavior we can calculate some measures we described in Chapter 3. We can for example

calculate the long-term average of customers in the system, since this is given by

$$E[L] = \sum_{n=0}^{\infty} np_n = \frac{\rho}{1-\rho}. \quad (5.83)$$

We obtain the mean sojourn time by applying Little's law to the mean number of customers in the system, which gives us

$$E[S] = \frac{E[L]}{\lambda} = \frac{\frac{1}{\mu}}{1-\rho}. \quad (5.84)$$

The mean number of customers waiting in the queue is given by

$$E[L^q] = \sum_{n=1}^{\infty} (n-1)p_n = \frac{\rho^2}{1-\rho}. \quad (5.85)$$

and the mean waiting time is thus giving by

$$E[W_q] = \frac{E[L^q]}{\lambda} = \frac{\frac{\rho}{\mu}}{1-\rho}. \quad (5.86)$$

This shows the power of the CTMC. We have found the limiting distribution of a general M/M/1 model and used it to determine a lot measures concerning the behavior of that queueing network. The CTMC approach is mainly good in mean value analysis of a queueing system. We can easily define the mean value of several other statistics.

It is even possible to derive the distribution of the sojourn time and waiting time of a customer. Let us define  $L_a$  as the number of customer in the system at the moment a new job arrives. Let  $B_k$  the service time of customer  $k$ . Thus  $B_1, B_2, \dots, B_k$  are an independent and exponentially distributed random variables with mean  $\frac{1}{\mu}$ . This also holds for the first customer, since the server times are exponential distributed and thus memoryless.

Let  $S$  denote the sojourn of a job, then

$$S = \sum_{k=1}^{L_a+1} B_k. \quad (5.87)$$

Thus we have for  $t \geq 0$

$$P(S > t) = P\left(\sum_{k=1}^{L^a+1} B_k > t\right) \quad (5.88)$$

$$= \sum_{n=0}^{\infty} P\left(\sum_{k=1}^{L^a+1} B_k > t \mid L^a = n\right) P(L^a = n) \quad (5.89)$$

$$= \sum_{n=0}^{\infty} P\left(\sum_{k=1}^{n+1} B_k > t\right) P(L^a = n). \quad (5.90)$$

$$= \sum_{n=0}^{\infty} \sum_{k=0}^n \frac{(\mu t)^k}{k!} e^{-\mu t} P(L^a = n) \quad (*) \quad (5.91)$$

$$= \sum_{n=0}^{\infty} \sum_{k=0}^n \frac{(\mu t)^k}{k!} e^{-\mu t} (1 - \rho) \rho^n \text{ by the PASTA property} \quad (5.92)$$

$$= \sum_{k=0}^{\infty} \sum_{n=k}^{\infty} \frac{(\mu t)^k}{k!} e^{-\mu t} (1 - \rho) \rho^n \quad (5.93)$$

$$(5.94)$$

$$= \sum_{k=0}^{\infty} \frac{(\mu \rho t)^k}{k!} e^{-\mu t} \quad (5.95)$$

$$= e^{-\mu(1-\rho)t} \quad (5.96)$$

(\*) Here we used  $\sum_{k=1}^{n+1} B_k$  is  $Erlang_{n+1}$  distributed. The  $Erlang_k$  distribution can be easily defined. Let  $X_1, \dots, X_k$  be a sequence of independent exponential distributions with mean  $\mu$ . Then  $Y = X_1 + \dots + X_k$  is an  $Erlang_k$  distributed random variable.

Therefore we conclude that the Sojourn time of a job is a exponential distributed random variable with parameter  $\mu(1-\rho)$ . Similar to this reasoning we can show that the waiting time of a job is 0 with probability  $(1-\rho)$  and is exponentially distributed with parameter  $\mu(1-\rho)$ . So we can use the CTMC approach even for the calculation the distribution function.

### 5.3.1.1 Processor sharing

Let us have a look at the M/M/1-PS queue. New customers arrive at rate  $\lambda$  and customers are served at rate  $\mu$ . Let us construct a CTMC for the number of customers in the system. It turns out that the state space and transition matrix of a M/M/1 queue and a M/M/1-PS queue are the same! Therefore we can immediately conclude that

$$p_n = (1 - \rho) \rho^n$$

is the limiting distribution of the CTMC. Also the mean value of customers in the system ( $E[L] = \frac{\rho}{1-\rho}$ ) and the  $E[S] = \frac{1}{1-\rho}$  are the same for the M/M/1 and the M/M/1-PS queue.

If we have a look at the CTMC corresponding to the number of customer in the system of a M/M/1-LPS queue, we will notice that also that CTMC is the same. It turns out that it does not matter in which order customer will be served. The mean number of customers in the system and the mean time that a customer spends in the system will always be the same. That does not mean that each system behave the same. Only the mean values are the same. A CTMC does not keep track of each job individually. It only keeps track of the number of costumers in the system and how this number behaves over time. Using CTMC's is great to analyze mean values, but less useful for gathering statistics about a single job. For example, it is really hard to apply the concept of time to live in a CTMC.

### 5.3.1.2 Other simple queueing networks

Another disadvantage of the approach of a CTMC is that not all queueing networks can be modeled as CTMC. A queueing network such as the  $M/G/1$  cannot be modeled as CTMC, since a general distribution does not always fit in a CTMC. At a first glance only M/M/c/n queueing networks (with possible some different server disciplines) can be modeled as CTMC. This is not the case, but therefore we need some trick. Let us have a look at the M/ $E_r$ /1 queue, where  $E_r$  is the Erlang distribution with  $r$  phases. If we try to construct a CTMC for the number of customers in the system, this will fail. The trick is to look at the number of phases to be done. An arrival will not count as 1 extra customer, but as  $r$  extra fases. This will give us the following balance equations:

$$\lambda p_0 = \mu p_1 \quad (5.97)$$

$$(\lambda + \mu)p_n = \mu p_{n+1} \text{ for } n = 1, \dots, r - 1 \quad (5.98)$$

$$(\lambda + \mu)p_n = \lambda p_{n-r} + \mu p_{n+1} \text{ for } n \geq r \quad (5.99)$$

$$\sum_{n=0}^{\infty} p_n = 1 \text{ (normalization equation)}. \quad (5.100)$$

Solving this system is harder. Let us assume we have solved this system of linear equation. We want to compute the mean number of customers in the system. Let  $q_n$  be

the probability that there are  $n$  customers in the system. This probability is equal to

$$q_0 = p_0 \quad (5.101)$$

$$q_n = \sum_{i=r(n-1)+1}^{rn} p_n \text{ for } n \geq 1 \quad (5.102)$$

and thus the mean number of customers in the system is

$$E[L] = \sum_{n=0}^{\infty} nq_n. \quad (5.103)$$

$$(5.104)$$

Also the mean value of the waiting time, mean value of the sojourn time, the mean value of number of customer in the queue can be easily calculated.

Therefore using CTMC for queueing models are not limited to only some M/M/1 queueing. There is a broader class of queueing models for which this approach is useful. Therefore it is an useful technique to solve queueing networks.

### 5.3.2 More complex queueing networks

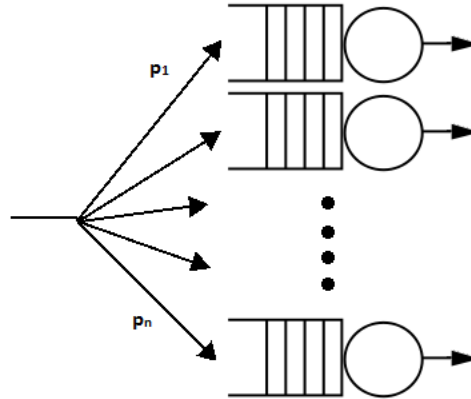
The approach of CTMC is not only limited to small queueing networks. This approach can also analyze more complex queueing networks with more than one queue or server.

#### 5.3.2.1 n M/M/1 queues in parallel

Let us have a look at a queueing network consisting of  $n$  M/M/1 queues in parallel. Jobs arrive at rate  $\lambda$  and go to queue  $i$  with probability  $x_i$ . The server rate of the  $i$ -th server is given by  $\mu_i$ . Figure 5.1 shows this queueing network.

By splitting property of the Poisson process we already know that the arrival process of queue  $i$  is again a Poisson process with parameter  $x_i\lambda$  and this process is independent of the arrival processes at other queues. Therefore this system can also be described as  $n$  independent M/M/1 queues with arrival rate  $x_i\lambda$  and server rate  $\mu_i$ . This results is not only valid for the M/M/1 queue, but for every queueing system with Poisson arrivals.

We already had a look at the M/M/1 queue and we can use the results to analyze this queueing system. This system is stable and has a limiting distribution if all individually M/M/1 queues are stable, and thus  $0 < \rho_i = \frac{x_i\lambda}{\mu_i} < 1$  for all  $i \in \{1, 2, \dots, n\}$  The mean number of customers in the system is equal to the sum of the mean number of customers




---

FIGURE 5.1: A queueing model of  $n$  parallel M/M/1 queues as described above.

of a single M/M/1 queue and thus is given by

$$E[L] = \sum_i E[L_i] = \sum_i \frac{\rho_i}{1 - \rho_i} \quad (5.105)$$

and applying Little's law yields

$$E[S] = \frac{E[L]}{\lambda} = \frac{1}{\lambda} \sum_i \frac{\rho_i}{1 - \rho_i}. \quad (5.106)$$

We can easily change the server discipline or the server time distribution. The only condition will be that the server time distribution of server  $i$  must be independent of all other servers and this must hold for all  $i$ .

### 5.3.2.2 $n$ M/M/1 queues in series

Let us again look at a queue network containing of  $n$  M/M/1 queues, this time all in series. Let  $\lambda$  be the arrival rate and let  $\mu_i$  be the server rate of server  $i$ . In order to analyze this system, we need a lemma.

**Theorem 5.29** (Burke's Theorem). *Consider a M/M/1, M/M/c or M/M/ $\infty$  queueing system in steady state behavior with arrival rate  $\lambda$  and server rate  $\mu$ . Then*

1. the departure process is a Poisson process with parameter  $\lambda$  and
2. the number of customers in the system is independent of the departure rate.

*Proof.* The idea of the proof is based on a time reversible CTMC. Let us assume that the corresponding CTMC of a M/M/1, M/M/c or M/M/ $\infty$  queueing system is time reversible. Thus the behavior of such system forward in time and backward in time is the same. Since forward and backward in time behave the same, the arrivals in the backward in time process are also a Poisson process with parameter  $\lambda$  and independent of the number of customers in the system. An arrival when looking backward in time in a CTMC corresponds with a departure in the forward CTMC. Therefore the departure process is also a Poisson process with parameter  $\lambda$  and the number of customer in the system is independent of the departure rate.

Let us now prove the assumption that the corresponding CTMC of a M/M/1, M/M/c or M/M/ $\infty$  queueing system is time reversible. We will only prove this for the M/M/1 queue. The prove for the M/M/c and M/M/ $\infty$  are similar. We have already shown that a CTMC is time reversible if  $p_i q_{ij} = p_j q_{ji}$ . Therefore we will have to check this condition for the M/M/1 queue. If  $i = j$  this statement is trivial. If  $|i - j| \geq 2$  then  $q_{ij} = 0 = q_{ji}$  and thus this statement is also trivial. Therefore we only consider the case  $i = j + 1$ . Then it follows that

$$p_i q_{i(i+1)} = p_{i+1} q_{(i+1)i} \Leftrightarrow \quad (5.107)$$

$$(1 - \rho) \rho^i \lambda = (1 - \rho) \rho^{i+1} \mu. \quad (5.108)$$

Since  $\rho = \frac{\lambda}{\mu}$  it follows that both expressions are equal. Thus the CTMC is time reversible.  $\square$

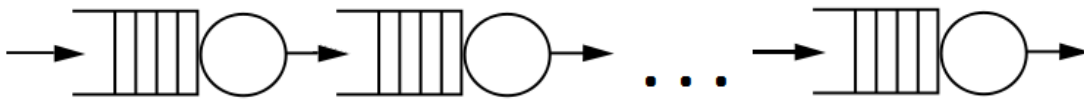


FIGURE 5.2: A queueing model of  $n$  M/M/1 queues in series as described above.

Since the idea of the proof is based on time reversible CTMCs, this result can be extended to more queueing networks. We only assume that the CTMC that models the number of customers in the system is time reversible, and we assume Poisson arrivals with parameter  $\lambda$ .

Let us return to the  $n$ -M/M/1 queues in series model. Let us define  $1 > \rho_i = \frac{\mu_i}{\lambda_i}$ . Since the departure distribution at server  $i$  is equal to the arrival distribution of server  $i + 1$  we can say, based on Burke's theorem that the arrival rate at each server is  $\lambda$  and this arrival distribution is independent of the previous servers. Therefore we can analyze each server individually. The mean number of customers in this queueing network is equal to the sum of the mean number of customer at a single server (in service and in the queue). Since all queues are M/M/1 queues, we already know their behavior.



Therefore we have

$$E[L] = \sum_{i=0}^n \frac{\rho_i}{1 - \rho_i} \quad (5.109)$$

and the mean sojourn time of a customer is giving by

$$E[S] = \sum_{i=0}^n \frac{\frac{1}{\mu_i}}{1 - \rho_i}. \quad (5.110)$$

Note that this approach does not only holds for M/M/1 queues in series, but for every queue which can be described by a time reversible Markov chain.

### 5.3.3 Even more complex networks

We can combine the results of queueing in series and parallel to analyze ever more complex queueing networks. Consider therefore the following queueing network consisting of three servers as described in Figure 5.3. We can analyze this queueing network as follows.

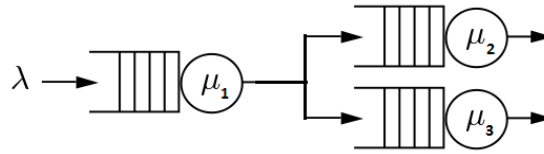


FIGURE 5.3: A queueing system containing three M/M/1 queues.

The departure process of the first server is a Poisson process with parameter  $\lambda$  which is independent of the number of jobs at the first server. Therefore we can say that the arrival rate at server 2 is equal to  $p_1\lambda$  and the arrival rate at server 3 is  $(1 - p_1)\lambda$  and server 2 and 3 are independent. Therefore we can analyze the three servers individually and combine the result. Since all three servers are M/M/1 queues, we obtain

$$E[L] = \frac{\rho_1}{1 - \rho_1} + p_1 \frac{\rho_2}{1 - \rho_2} + (1 - p_1) \frac{\rho_3}{1 - \rho_3} \quad (5.111)$$

and

$$E[S] = \frac{1}{\lambda} E[L] \quad (5.112)$$

where  $\rho_1 = \frac{\lambda}{\mu_1}$ ,  $\rho_2 = \frac{p_1\lambda}{\mu_2}$  and  $\rho_3 = \frac{(1-p_1)\lambda}{\mu_3}$ . In order to get a stable queueing network, we need  $0 < \rho_i < 1$  for  $i = \{1, 2, 3\}$ .

We can even extend this model much further. We know how to solve some queues in

parallel or in series. Therefore we can add as many of them to the model, and we can still analyze the model with the CTMC approach by looking at each queue individually and combining the results to analyze the complete model. Note that we do not allow loops in the queueing model. That means, if a customer is served at server  $i$  the customer cannot go back to server  $i$ .

## 5.4 Solving the CreateOrder queueing network

It is time to have a look at the CreateOrder services. We will construct some queueing network and try to analyze this network via CTMC. In our analysis, we make the following assumptions:

1. The arrival process is a Poisson process with parameter  $\lambda$
2. The initiate order handler is deployed at four different nodes with the server discipline limiting processor sharing. A node can handle  $s_1$  jobs at the same time.
3. The server times of a node of the initiate order handler are independent and exponentially distributed with parameter  $\mu_1$ .
4. All four nodes of the initiate order handler uses his own queue with queue length  $r_1$ .
5. A new arrival goes to the queue of node  $i$  with equal probability for every node and independent of other arrivals.
6. After the initiate order handler, the process continues to the same queue at PIM with queue length  $r_2$
7. PIM is deployed on one node with the server discipline limiting processor sharing. A node can handle  $s_2$  jobs at the same time. The server times are exponentially distributed with parameter  $\mu_2$ .
8. After PIM, the process continues to the same queue at AOC with queue length  $r_3$ .
9. AOC is deployed on one node with the server discipline limiting processor sharing. A node can handle  $s_3$  jobs at the same time. The server times are exponentially distributed with parameter  $\mu_3$ .
10. After AOC, the process is randomly split into the four nodes of the OR. Each node has the same probability of being chosen and this splitting is independent of other jobs.

11. All four nodes of the initiate order handler uses a different queue with queue length  $r_4$
12. The server discipline of a node of the OR is a limiting processor sharing. A node can handle  $s_4$  jobs at the same time. The server times of a node of the OR are independent and exponentially distributed with parameter  $\mu_4$ .
13. We do not include sessions or time to live.

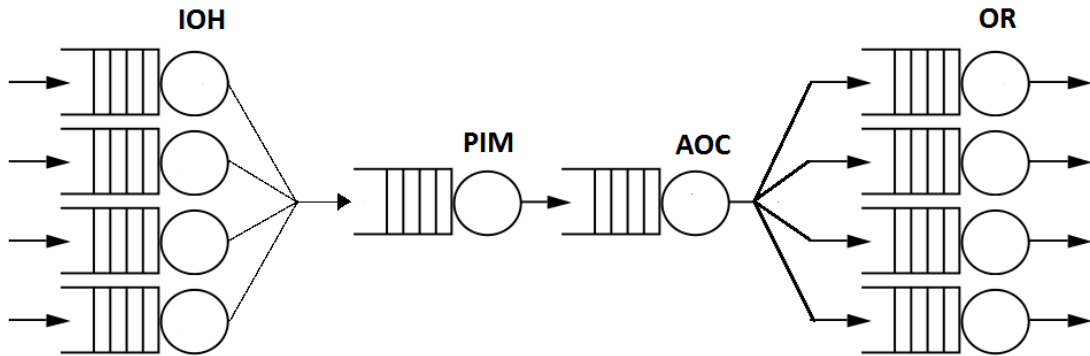


FIGURE 5.4: The queueing network corresponding with the CreateOrder service.

Figure 5.4 shows us the queueing network based on these assumptions. We will try to analyze this queueing network under different loads (different values of  $\lambda$ ). We will use the following parameters, which are measures by the ING Bank:

1.  $\mu_1 = \frac{1}{0.006}$
2.  $\mu_2 = \frac{1}{0.018}$
3.  $\mu_3 = \frac{1}{0.013}$
4.  $\mu_4 = \frac{1}{0.065}$
5.  $r_1 = 250$ . Normally there would be one queue for all four nodes of 1000. Since we split the queue in four different queues, we divide the queue length over all queues.
6.  $r_2 = 60 - s_2$ . Normally it would be 1000. Due to the sessions it cannot be higher than  $60 - s_2$ .
7.  $r_3 = 60 - s_3$ . Normally it would be 1000. Due to the sessions it cannot be higher than  $60 - s_3$ .
8.  $r_4 = 15 - s_4$ . Normally it would be infinity. Due to the sessions it cannot be higher than  $15 - s_4$ .

9.  $s_1 = 15$ .
10.  $s_2 = 60$ . Normally it would be infinity. Due to the sessions it cannot be higher than 60.
11.  $s_3 = 4$ .
12.  $s_4 = 5$ .

Using this information we can conclude that each node in our queueing network for the CreateOrder module can be described as  $M/M/1/(r+s) - PS$  queue with parameters  $\lambda$  (arrival rate),  $\mu$  (server rate),  $r$  (queue length) and  $s$  (number of jobs a server can handle). If we can show that the number of customers in a  $M/M/1/(r+s) - PS$  queue can be modeled as time reversible CTMC, we can use the earlier discussed tools.

The state space for such process is given by  $\{0, 1, 2, \dots, r+s\}$ . The corresponding transition matrix  $Q$  is a  $(t+s+1) \times (t+s+1)$  matrix and is given by

$$Q = \begin{pmatrix} -\lambda & \lambda & & & & & \\ \mu & -(\lambda + \mu) & \lambda & & & & \\ & \mu & -(\lambda + \mu) & \lambda & & & \\ & & \mu & -(\lambda + \mu) & \lambda & & \\ & & & \mu & -(\lambda + \mu) & \lambda & \\ & & & & & \ddots & \\ & & & & & \mu & -\mu \end{pmatrix}. \quad (5.113)$$

The balance equations are therefore given by

$$\lambda p_0 = \mu p_1 \quad (5.114)$$

$$(\lambda + \mu)p_n = \mu p_{n+1} + \lambda p_{n-1} \text{ for } n \in \{1, 2, \dots, r+s-1\} \quad (5.115)$$

$$\sum_{n=0}^{r+s} p_n = 1 \text{ (normalization equation)}. \quad (5.116)$$

Analogue to the  $M/M/1$  model this will give us

$$p_n = \rho^n p_0 \text{ for } n \in \Omega \quad (5.117)$$

$$\sum_{n=0}^{r+s} p_n = 1 \quad (5.118)$$

and thus

$$p_n = \rho^n \frac{\rho - 1}{\rho^{r+s+1} - 1}. \quad (5.119)$$

This CTMC is time reversible. For the proof, we have to check that  $p_i q_{ij} = p_j q_{ji}$ . This proof is analogue to the proof that the CTMC corresponding to the M/M/1 queue is also time reversal. Therefore the departure process of a  $M/M/1/(r+s) - PS$  queue is also a Poisson process. Note that not all arrivals enter the queue of the  $M/M/1/(r+s) - PS$  queue. Some jobs are blocked. By the Pasta property the blocking probability is equal to the limiting probability that the system is full, which is  $p_{r+s}$ . Therefore the departure process has a rate of  $\lambda p_{r+s}$ .

Let us return to the model of the CreateOrder flow.

The expressions for mean value analysis are not really nice. Therefore we will use

$$E[L] = \sum_{n=0}^{r+s} n p_n \text{ and} \quad (5.120)$$

$$E[S] = \frac{E[L]}{\lambda} \quad (5.121)$$

in our calculations. Most computational software program can handle these expressions easily.

We have earlier seen how to analyze these kind of complex networks. Based of this theory, I constructed a Mathematica file to see the behavior under certain loads. The Mathematica-file can be viewed in [Appendix A](#).

### 5.4.1 Bottleneck analysis

In order to analyze this model we will have a look at some statistics of the behavior of the model under certain loads. In the first place we will have a look at the blocking probability and the throughput. The blocking probability  $B$  is given by

$$B = 1 - (1 - B_1) * (1 - B_2) * (1 - B_3) * (1 - B_4) \quad (5.122)$$

where  $B_i$  is the blocking probability at stage  $i$  of the process. So 1 stands for IOH, 2 for PIM, 3 for AOC and 4 for OR. We already noticed that by the Pasta property the blocking probability is equal to the limiting probability that the system is full. The throughput  $TP$  is given by

$$TP = (1 - B)\lambda \quad (5.123)$$

where  $\lambda$  is the arrival rate. I.e. the throughput is the rate of non-blocking jobs.

In [Figure 5.5](#) you can see that most of the jobs are blocked at PIM. Therefore it look like PIM will be the bottleneck of this system. The throughput suggests that there is

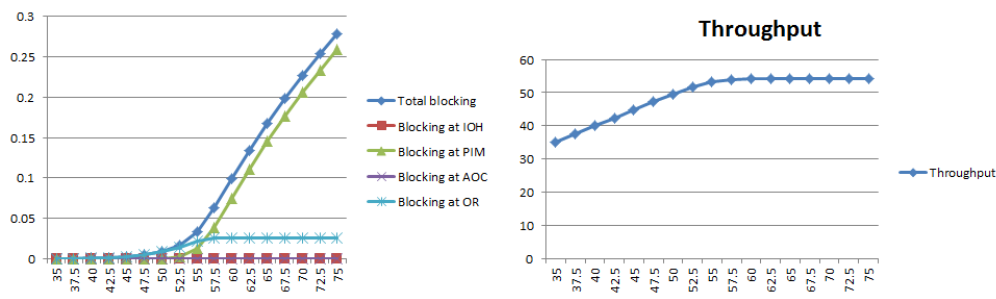


FIGURE 5.5: Blocking probability and throughput in the current situation

almost no drop off until PIM cannot handle all jobs. From a mathematical point of view, this is the case of  $\rho \geq 1$ . This is around a load of 55.5. Due to the fact that we do not have infinite queue lengths the system can handle a little bit lower. The maximum throughput is therefore 54.1. Figure 5.6 shows the blocking probability under certain

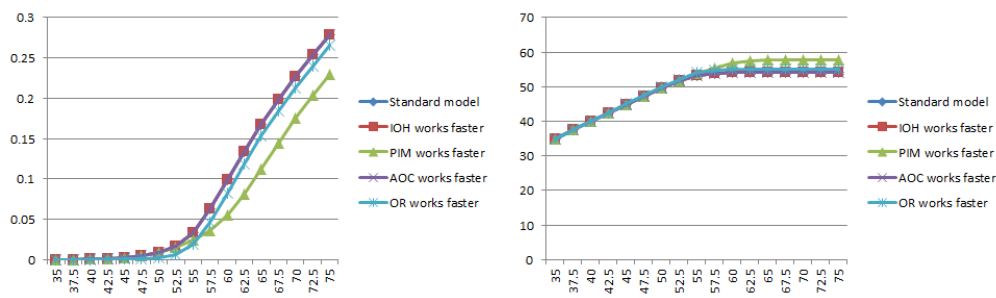


FIGURE 5.6: Blocking probability and throughput under some scenarios

scenarios. The standard model is just the model described above. The other four models are models where we increase the work speed of a single server with 10%. We obtain almost no progress if we increase the IOH or AOC. It seems they can handle all the work still easily. The performance of the system will increase if we increase the work speed of PIM. Since we already stated that PIM is a bottleneck, it is not surprising.

It is more surprising that the performance of the system also increases a bit by increasing the work speed of OR. But this is not that strange. If we compare the  $\rho$  for a OR-server, it is also close to 1 already. Since OR is close to being the bottleneck, only increase the work speed at PIM, do not increase the throughput a lot. Another scenario is to add more servers. The results of this are shown in figure 5.7. These graphs tell us the same story. PIM is the bottleneck, but OR is close to be the bottleneck. It is even more clear if we look at which step in the process gives the blocking. If we have a look at the scenario where we added another PIM, deployed two nodes, almost all blocking comes from OR! Figure 5.8 shows these result.

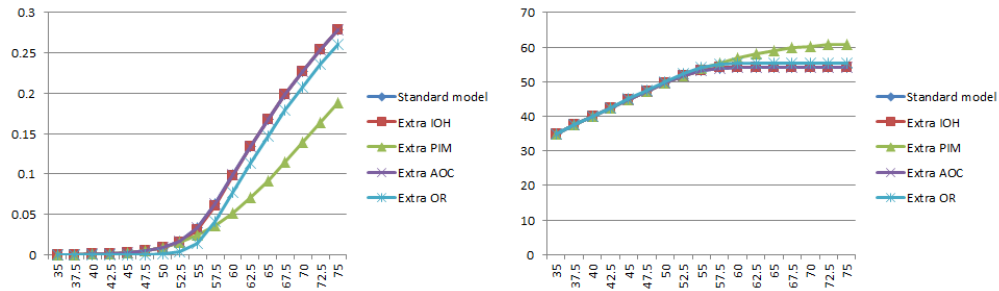


FIGURE 5.7: Blocking probability and throughput under some other scenarios

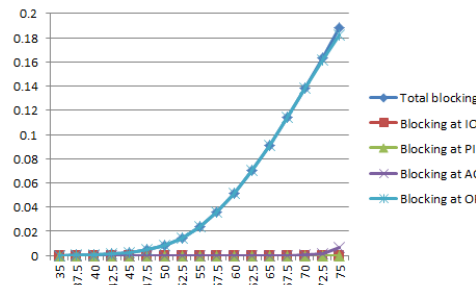


FIGURE 5.8: Blocking probability after adding an extra PIM-server.

Let us also have a look at sessions. We did not include sessions in the CTMC. But we can say something about the influence of sessions. Sessions become a problem when there are too many jobs in the system, excluding the jobs waiting in a queue of IOH. Figure 5.9 shows us that there are mostly no jobs waiting in the queue of IOH, since average number of jobs is really low. Therefore we can use the mean number of jobs in the system as measure for the mean number of used sessions. If the mean number of used sessions is larger than 60 (the number of sessions), the system became unstable because of the lack of sessions. Therefore we are interested at which load the mean number of

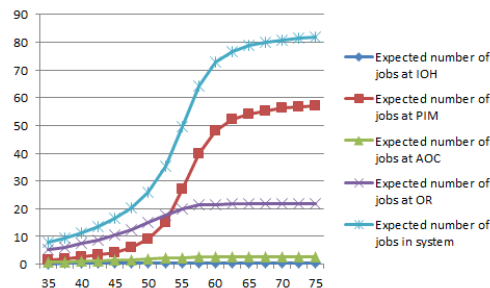


FIGURE 5.9: Expected number of customers.

jobs in the system exceed 60. Based on the graph we can see that this would be the case from a load around 57. It seems that the number of sessions becomes problematic

because PIM cannot handle his work. All used jobs are waiting in the queue of PIM and use sessions. It is not the case that  $4 \times 15$  sessions is too few. Let us therefore double the number of sessions, so we have  $4 \times 30$  sessions. Figure 5.10 shows us the results. The

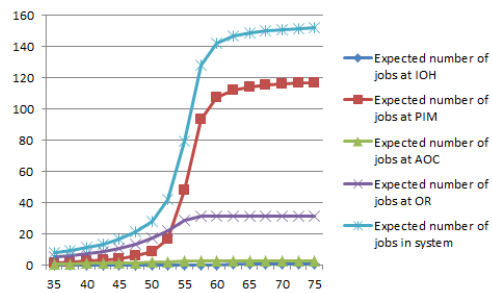


FIGURE 5.10: Expected number of customers after doubling the sessions.

number of sessions will be again problematic around a load of 57. Therefore we may conclude that the number of sessions is not the bottleneck. It is just PIM that cannot handle all jobs and thus lots of jobs are waiting in the queue and eat up the number of sessions.



## Chapter 6

# Comparison

We have tried to model the CreateOrder module with two different approaches. We will use this chapter to compare the two modules. We will discuss the cons and pros of both approaches. We will start with the simulation.

The main difference between both approaches can be describes as details versus calculation speed. The simulation is detailed, but has a slow calculation speed. The analytical approach has a much higher calculation speed, but is less detailed.

A simulation is a very useful tool to analyze complex systems. But it takes a lot of time to construct such a model, run the model and analyze the model. Another drawback of a simulation is the lack of flexibility. You analyze the model under a certain set of parameters. For any other set of parameters, you have to run the simulation again. Each set of parameters takes a lot of time to analyze. Therefore a simulation is mostly useful if you have a lot of time.

At first glance, Rockwell Arena looks like a really good and easy to use program. It works very intuitive. But the options are pretty limiting. Therefore it is maybe a good idea to look at other simulation programs. Programs like Matlab, Octave, R and Mathematica can all be used to simulate queueing networks. The drawback of these programs is that you have to build up the simulation from scratch. Rockwell Arena has a box in which you can specify the arrival pattern. In the other programs you have to write your own module for the arrival pattern. Rockwell Arena gathers its own statistics and has some boxes to create extra statistics, in the other program you have to write your own modules to gather statistics. Creating your own modules take some time, but this will result in a better tool to analyze a queueing networks through simulating. This makes the simulation approach more powerful.

Trying to construct processor sharing was really a time consuming aspect. Therefore it is a shame that Rockwell Arena cannot handle processor sharing which is used very often in queueing networks of ICT landscapes. We have attempted different approaches

to model processor sharing in Rockwell Arena, but none of them worked due to how Rockwell Arena updates the processing times. There was only one solution to solve this problem and that is by writing a Java script which communicates with Rockwell Arena and changes the way how Rockwell Arena updates all variables. Due to this Java script the simulation runs very slow. And since simulation is already a time consuming approach, it would be even more time consuming. Therefore this solution is not really a proper solution.

Since we could not use the M/M/1-PS queue, we have used the M/M/1 queue. We have seen that these queues have a lot in common, but there are also some differences. The mean time in the system is the same, but the real distribution is different. Applying the concept of Time to live on the M/M/1-PS and M/M/1 model lead to two different results, since the distributions are different. Therefore the results of the simulation would improve if we had used a server which can handle processor sharing.

An analytic approach like our Markov theory approach does not have all the drawbacks of a simulation. When the theory behind the analytic approach is known, it is in many cases easy to construct the model. Since the approach is analytic we want to solve the model for a general set of parameters. Evaluating the results for some specific set of parameters is mostly easy if the general solution is known. The drawback of the analytical approach is that not all systems have an analytical solution and thus not every system can be solved analytically. Therefore a model for the analytic approach has mostly less details. If we have a look at our approach to model the CreateOrder, we had to make a lot of assumptions, which made the system easier. Many complex systems cannot be analyzed analytically, since we do not have the tools to analyze such complex networks. This brings us to the next drawback of an analytical approach. Solving a complex problem analytically requires some mathematical framework which can be used to solve the problem analytically. In order to solve the model of the CreateOrder analytically, we had to come up with all the Markov theory. Such framework is required to solve a problem analytically.

Therefore it is good to have both approaches. With the Markov theory we can quickly determine the possible bottleneck. We know where the bottlenecks probably are. With that knowledge we can search a bit deeper with the simulation. We can immediately focus on sets of parameters near the bottleneck of the Markov approach. So we can combine the advantages of both approaches together. We are able to find the bottleneck with the Markov approach and analyze the bottleneck in more details with the simulation. Finding the bottleneck in short term, the Markov approach could be enough. But for a really detailed simulation which will fix the bottleneck on the long term, one has to combine the analytic approach with the simulation.

## 6.1 Comparing the results

We already noticed some pros and cons of both approaches in modeling the Create-Order module. It takes a while to get some useful results from the simulation. And these results are only valid for some specific set of parameters. Is it almost not done to determine the influence of all parameters in each setting. On the other hand, the Markov theory approach was very quick and flexible enough to determine the performance of the system under a lot of sets. We even added another server to the model in the Markov theory approach and we had the results immediately. If we want to do the same for the simulation, we had to rebuild the whole simulation, and simulate the new system again under a lot of scenarios. That will take a lot more time!

On the other hand, it was impossible to add sessions and time to live to the Markov approach. Based on our model we tried to guess the influence of sessions, but the influence of time to live remains a mystery. If we compare the results of both models, we can conclude that both give on average the same idea of the bottleneck, which must be at PIM. But we see a huge difference if we look at the influence on sessions. Based on the Markov approach we conclude that adding more sessions had almost no influence on the system. Maybe the influence was a bit positive, but the difference was almost negligible. If we had a look at the simulation, adding the same number of sessions had a huge influence on the system, in fact a negative influence. We can explain the difference if we had a look at the time to live. More sessions lead to more jobs in the queue of PIM and thus more timeouts at PIM.

On average the behavior was the same. Both approaches pointed to PIM as the bottleneck. In both approaches, the bottleneck starts to play a role if the load is around 50-55 TPS. In both cases, PIM can handle at most 55 jobs per second on average. Increasing the work speed of the PIM had a positive influence on the performance of the system in both cases.



# Chapter 7

## Final remarks

### 7.1 Conclusion

In order to help the ING Bank by analyzing a part of the ICT landscape of the Bank, we have modeled the CreateOrder services. After an introduction of the LOM and the CreateOrder service in Chapter 2, and an introduction of the mathematical concept of queueing networks in Chapter 3, we had a look at the first approach to model the CreateOrder service. This approach uses a simulation and is explained in Chapter 4. For this approach we used the program Rockwell Arena. We have developed some tools to construct the CreateOrder service in the program. We had a look at modeling a M/M/1 queue, a finite queue length, sessions and time to live. Unfortunately, that we could not model the server discipline processor sharing in Rockwell Arena and therefore conclude in Chapter 6 that it is maybe a good idea to look at other programs that can be used for simulation.

Nevertheless, we were able to construct a model for the CreateOrder service. We compared the set of parameters acquired from the ING under different loads with some adapted sets of parameters. This gives us a broader picture of the bottleneck. All our analysis on the simulation lead us to the PIM-server. That must be the bottleneck of the CreateOrder flow.

Our next step was an analytic approach using results of (Continuous-Time) Markov chains. This approach is described in Chapter 5. We started this Chapter with the definition of a Markov Chain and tried to figure out when the stationary distribution exist. We discovered that a Markov Chain has a unique stationary distribution if the Markov Chain is irreducible and aperiodic and the state space is finite. A Markov chain on a countable state space has a unique stationary distribution if the Markov chain is irreducible, aperiodic and positive recurrent. We also had a look at a time reversible

Markov chain.

From the Markov chain we made a jump to the Continuous-time Markov chain (CTMC). We have also had a look at the limiting distribution of the CTMC. We were mainly interested in how to calculate the limiting distribution. We discovered two approaches: Solving the balance equation or look at the Embedded Markov chain. We also had a look at a time reversible CTMC.

After this introduction we made our way to queueing theory. We discovered that some queueing networks can be seen as CTMC. Solving the limiting distribution of a CTMC is the key to analyzing this kind of queueing network. With the use of the limiting distribution, we can determine for example the average sojourn time of a customer and the mean number of customers in the system. It was even possible to solve the distribution of the waiting and sojourn time of a customer in a M/M/1 model!

There are also larger queueing networks that can be described as a CTMC. Since finding the limiting distribution of a CTMC is not always that easy, we developed a theory to solve the limiting distribution of larger queueing networks. This theory is mainly based on Burke's theorem (Chapter 5) and some properties of the Poisson process (discussed in Chapter 2). This allowed us to solve every queueing system with Poisson arrival and without a loop if each server individually can be solved as a CTMC. That means, if a job is processed at server  $i$ , it is impossible to return back to server  $i$  for that specific job and this must hold for all servers.

With this result we made a model for the CreateOrder service. It was not possible to include sessions and time to live. We put the results in a Mathematica file (See Appendix A) so we can easily evaluate different sets of parameters. This allowed us to easily compare some scenarios to analyze the model. From the analysis it follows that the PIM-server was the bottleneck, which was also the bottleneck for the first approach.

## 7.2 Future research

Although both approaches lead to the conclusion that PIM is the bottleneck for the CreateOrder service, we can not conclude that PIM is the bottleneck in the complete system. There are many other processes in the LOM. Some of them use PIM, other do not use PIM. To get a conclusion for the complete system, more services must be modeled. This thesis can be used as a guide for this framework. Most of the services can be modeled the same way. But some processes will behave a bit different and thus you will hit some problems that are not discussed in these thesis.

The challenge for the simulation lies mostly in the fact of adding the server discipline processor sharing to the simulation. This cannot be done in Rockwell Arena. Therefore I will suggest to look for another program that can run the simulations. In Chapter 6, I

already mentioned Matlab, Octave, R and Mathematica are suited programs for running a simulation, although some might be better suited than others. None of these programs have a drop-and-drop modules like Rockwell Arena with the pre-defined modules used in queueing network.

There is more future research in the area of the Markov chains. For example: How to deal with batch arrivals. In the CreateOrder service there were no batch arrival, but in other processes there are batch arrivals. Another question is how to deal with Sessions? Now we dealt with sessions by looking at the average number of customers within the sessions. If this is less than the number of sessions, we assumed that the sessions are not all used. If the average number of customers is greater than the number of sessions, we concluded that the number of sessions was too small and thus the queue at the front of the sessions will grow to infinity. But maybe there is a way to include sessions immediately in the model.

The same holds for time to live. Since a continuous-time Markov chain keeps track of the global flow through the network and does not keep track of each individual job, it is hard to add time to live to the model. The trick we used for the simulation, to let the processing time be the minimum of some random variable and the time until the time to live exceeded will not work in the case, since this is conflicting the Markov property. This brings us to another question: Can we say anything about the system if the arrivals are not Poisson distributed or the server times not exponentially distributed? At this stage it was already difficult to get the mean processing times of the servers.





# Appendix A

## Mathematica code

This chapter contains the Mathematica used for the Markov model of the CreateOrder module.

```
lambda1:={35, 37.5, 40, 42.5, 45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5, 75}
```

```
lambda2:=(1 - B1) * lambda1
```

```
lambda3:=(1 - B2) * lambda2
```

```
lambda4:=(1 - B3) * lambda3
```

```
mu1:=1/0.006
```

```
mu2:=1/0.018
```

```
mu3:=1/0.013
```

```
mu4:=1/0.065
```

```
m1:=4
```

```
m2:=1
```

```
m3:=1
```

```
m4:=4
```

```
rho1:=lambda1/m1/mu1
```

```
rho2:=lambda2/m2/mu2
```

```
rho3:=lambda3/m3/mu3
```

```
rho4:=lambda4/m4/mu4
```

```
q1:=250
```

```
q2:=s - s2
```

```
q3:=s - s3
```

**q4:=s1 - s4**

**s1:=15**

**s2:=4**

**s3:=s**

**s4:=5**

**s:=s1 \* m1**

**p1[x1\_]:=rho1^x1 \* (rho1 - 1)/(rho1^(q1 + s1 + 1) - 1)**

**p2[x5\_]:=rho2^x5 \* (rho2 - 1)/(rho2^(q2 + s2 + 1) - 1)**

**p3[x6\_]:=rho3^x6 \* (rho3 - 1)/(rho3^(q3 + s3 + 1) - 1)**

**p4[x7\_]:=rho4^x7 \* (rho4 - 1)/(rho4^(q4 + s4 + 1) - 1)**

B1 = blocking at 1 (IOH)

B2 = blocking at 2 (PIM)

B3 = blocking at 3 (AOC)

B4 = blocking at 4 (OR)

B = total blocking

TP = throughput

**B1 = N[rho1^(q1 + s1) \* (rho1 - 1)/(rho1^(q1 + s1 + 1) - 1)]**

**B2 = N[rho2^(q2 + s2) \* (rho2 - 1)/(rho2^(q2 + s2 + 1) - 1)]**

**B3 = N[rho3^(q3 + s3) \* (rho3 - 1)/(rho3^(q3 + s3 + 1) - 1)]**

**B4 = N[rho4^(q4 + s4) \* (rho4 - 1)/(rho4^(q4 + s4 + 1) - 1)]**

**B = 1 - (1 - B1) \* (1 - B2) \* (1 - B3) \* (1 - B4)**

**TP = (1 - B1)(1 - B2)(1 - B3)(1 - B4) \* lambda1**

E1= expected number of jobs at machine 1

E2= expected number of jobs at machine 2

E3= expected number of jobs at machine 3

E4= expected number of jobs at machine 4

Et = expected number of jobs in whole system

**E1 = m1 \* Sum[x1 \* p1[x1], {x1, 0, s1 + q1}]**

**E2 = m2 \* Sum[x2 \* p2[x2], {x2, 0, s2 + q2}]**

**E3 = m3 \* Sum[x3 \* p3[x3], {x3, 0, s3 + q3}]**

$$\mathbf{E4} = \mathbf{m4} * \mathbf{Sum}[\mathbf{x4} * \mathbf{p4}[\mathbf{x4}], \{\mathbf{x4}, \mathbf{0}, \mathbf{s4} + \mathbf{q4}\}]$$

$$\mathbf{Et} = \mathbf{E1} + \mathbf{E2} + \mathbf{E3} + \mathbf{E4}$$

R1= expected total time a job spends at machine 1

R2= expected total time a job spends at machine 2

R3= expected total time a job spends at machine 3

R4= expected total time a job spends at machine 4

Rt= expected total time a job spends in the whole system

$$\mathbf{R1} = \mathbf{Sum}[\mathbf{x1} * \mathbf{p1}[\mathbf{x1}], \{\mathbf{x1}, \mathbf{0}, \mathbf{s1} + \mathbf{q1}\}]/\mathbf{lambda2}$$

$$\mathbf{R2} = \mathbf{Sum}[\mathbf{x2} * \mathbf{p2}[\mathbf{x2}], \{\mathbf{x2}, \mathbf{0}, \mathbf{s2} + \mathbf{q2}\}]/\mathbf{lambda3}$$

$$\mathbf{R3} = \mathbf{Sum}[\mathbf{x3} * \mathbf{p3}[\mathbf{x3}], \{\mathbf{x3}, \mathbf{0}, \mathbf{s3} + \mathbf{q3}\}]/\mathbf{lambda4}$$

$$\mathbf{R4} = \mathbf{Sum}[\mathbf{x4} * \mathbf{p4}[\mathbf{x4}], \{\mathbf{x4}, \mathbf{0}, \mathbf{s4} + \mathbf{q4}\}]/((\mathbf{1} - \mathbf{B4}) * \mathbf{lambda4})$$

$$\mathbf{Rt} = \mathbf{R1} + \mathbf{R2} + \mathbf{R3} + \mathbf{R4}$$



# Bibliography

- [1] Ivo Adan and Jacques Resing. Queueing theory. *Lecture notes*, 2002.
- [2] Stephen C. Graves John D. C. Little. Building intuition: International series in operations research and management science. *Springer*, 2008.
- [3] John D. C. Littles. A proof for the queueing formula:  $l = \lambda w$ . *Operations Research, Volume 9, Issue 3*, pages 383–387, 1961.
- [4] Rockwell Arena. Getting started with arena. *Rockwell Arena*, 2010.
- [5] Elizabeth L. Wilmer David A. Levin, Yuval Peres. Markov chains and mixing times. *AMS*, 2008.
- [6] Henk C. Tijms. A first course in stochastic models. *Wiley*, 2003.
- [7] Sheldon M. Ross. Introduction to probability models 10th edition. *Academic press*, 2010.
- [8] László Szeidl László Lakatos and Miklós Telek. Introduction to queueing system with telecommunication applications. *Springer*, 2010.