Networks of Polling Systems

Author: Laurent van den Bos
Supervisor: dr.ir. Marko Boon
Abstract

When cars travel through a city, they may pass a few intersections. These intersections might be busy or not. In general one can assume that customers arrive at a network of intersections, modeled as a polling model, and follow a path in it. The properties of an intersection can trivially be translated to the parameters of a polling model. The main question in this report is whether one can predict if the customer can traverse the network in a finite time or not. Note that there is no assumption about the stability of the several intersections.

Although several methods exist to study the behaviour of a polling model, not all methods suit for this situation because in this situation the method has to work with unstable (or saturated) polling models. The buffer occupancy method can be used for its analysis and from this method a computer program can be derived to model the situation. Also, a simulation can be written to verify the model. In this report, both methods are used and results were obtained for several cases.

The interesting result is that an polling model that seems to be unstable can become stable in a network. This can happen if it is linked to an unstable polling model that has different departure intensities than arrival intensities.
## Contents

1 Introduction ........................................ 3  
   1.1 Structure ........................................ 3  
   1.2 Acknowledgements ................................ 3  

2 Problem description .................................. 4  
   2.1 Intersections ..................................... 4  
   2.2 Main problem ..................................... 5  

3 Literature Review .................................... 6  
   3.1 Polling models .................................... 6  
      3.1.1 Birth and Death process .................. 6  
      3.1.2 Polling Model ......................... 7  
      3.1.3 Networks of polling models ............ 7  
   3.2 Assumptions of the network .................... 8  
   3.3 Stability ........................................ 8  
   3.4 Fluid models ................................... 9  

4 Theoretical Analysis ................................ 11  
   4.1 Stable situation ................................. 11  
   4.2 One unstable polling model ................... 12  
   4.3 N queues ....................................... 14  
      4.3.1 The situation ............................ 14  
      4.3.2 Serving one queue ....................... 15  
      4.3.3 Serving a cycle ......................... 16  

5 Implementation ..................................... 17  
   5.1 Numerical implementation ..................... 17  
   5.2 Simulation ..................................... 18  
      5.2.1 Monte-Carlo simulations ............... 20  
      5.2.2 Discrete-event simulations ............ 21  
      5.2.3 A simple example ....................... 22  

6 Main results ......................................... 23  
   6.1 Fluid model .................................... 23  
      6.1.1 Stable situation ........................ 23  
      6.1.2 Two queues .............................. 23  
      6.1.3 N queues ................................ 23  
   6.2 Simulation ...................................... 24
Chapter 1

Introduction

Everybody hates waiting. Whether you go to the store, use your phone to call a helpdesk, or wait in front of a traffic light, you probably have to wait. Because of this, queues have been studied very well. Sometimes, several queues form an intersection and that is why intersections have also been studied very well.

One aspect that has not been studied very well yet is that of several linked intersections. This is the case in traffic, where one car travels around a city on several roads and crosses several intersections. These intersections influence each other. If one lane of an intersection has a green light and is linked to another intersection, this second intersection receives those cars after a while and the waiting times of the cars that were already present become larger (there are more cars that have to be served).

In this report, a network of those intersections is studied. Several intersections are linked together and influence each other. The main question will be whether it is possible to determine if a path through the network can be travelled fast or not. In other words: the influence of a busy intersection on another intersection is studied.

1.1 Structure

This report is globally divided in three parts. First, the problem is described in Chapter 2 and the mathematical principles needed for the analysis are introduced in Chapter 3. These mathematical principles have been studied before, hence this chapter consists of a small literature study.

The next part consists of Chapter 4 and Chapter 5. In these chapters, the network is analysed and several properties are derived. Also, algorithms to determine numerical results are discussed. Finally, a small simulation is described.

In the last part (Chapter 6 and Chapter 7) numerical results are presented and a conclusion is drawn from the results. Several suggestions for further research are also presented.

1.2 Acknowledgements

This report is the result of the work of a bachelor student. But without his supervisor this report would not be there. The supervisor has a special role during a project. He not only has a lot more knowledge than the student, he also makes sure that the student does not get stuck with unsolvable problems. The supervisor of this project did his job very well. That is why I, the writer, would like to thank Marko Boon for his great work during the last months.
Chapter 2

Problem description

First, the problem will be illustrated using a realistic example. After this, the main question will be formulated.

2.1 Intersections

Most of the cities in the Netherlands have a so-called ring around them. The ring is an arterial road for cars which can be used to travel around the city. An example of a ring can be found in Figure 2.1.

When a lot of people enter a city at one point (something that might happen every morning or evening), it might be possible that they create a traffic jam at that position. It might be possible that a traffic jam at the ring influences the whole traffic in the city. Hence the main question of this report is whether multiple intersections can influence each other, and quantify that relation.

Figure 2.1: City of Eindhoven

Source: HERE Maps
2.2 Main problem

Now it is possible to state the problem that will be discussed in this report. Suppose there are several connected intersections. An intersection is a place where several roads cross each other. If one lane has a green light, the cars at the other lanes have to wait.

When a car leaves the intersection, it may go to another intersection if there exists a road to that intersection. The main question in this report is whether it is possible to determine if the sojourn time in the network (the time difference between entering and leaving the network) has an upper limit. Note that the sojourn time is not explicitly computed. It is only determined whether the upper limit is finite.

In this report, it is assumed that customers (and not cars) enter the system. Note that this can also be boxes, bottles, animals or network packages, hence the results can be extended to more practical applications.
Chapter 3

Literature Review

3.1 Polling models

Before one even can talk about an intersection in a formal way, one must introduce that formally and determine a model for the intersection. An intersection can be made as complex as possible (for example with several lanes for cars, bikes and pedestrians), hence several simplifications have to be formulated. In this situation, the polling model can be used and is already studied very well.

In this section, the polling model will be formally introduced. As a polling model exists of several queues, also the queue must be introduced. For the latter one, the already known properties of the birth and death process can be used. This is a good point to start. Only the concepts important for this research are discussed.

More information about polling models in general can be found in [4, 9, 3].

3.1.1 Birth and Death process

Consider a system whose state is represented at any time by the number of customers in the system, and the amount of births and deaths are dependent of the amount of customers in the system. Suppose that when there are \( n \) \((n \in \mathbb{N}\), and \( 0 \leq n < \infty \)) customers in the system, new births happen at a rate \( \lambda_n \), where \( \lambda_n = 1/E[A_n] \). Here, \( A_n \) is the time between two births. Suppose that customers in the system die in the system at a rate \( \mu_n \), where \( \mu_n = 1/E[B_n] \). Here, \( B_n \) is the time between two deaths.

Such a system is called a birth and death process. The parameters \( \lambda_k \) and \( \mu_k \) are called the birth and death rate. The values of \( A_k \) and \( B_k \) can be determined from \( \lambda_k \) and \( \mu_k \), and are called the birth and the death time respectively.

A queue is a special case of a birth and death process (see [10]). The births of the birth and death process are the arrivals, the deaths are the customers leaving the system. Also, for the queues in this report it is assumed that all \( \lambda_n \) are equal (so just \( \lambda \)). The same holds for \( \mu_n \), which becomes just \( \mu \).

The same holds for the birth and death time. The birth time \( A \) is called the interarrival time and \( B \) is the service time of the customers. One can see a queue as a row with people waiting until it is their turn to be served. Customers arrive after each other, and the time between two arrivals is \( A \). The server (the person or thing that serves the customers) uses \( B \) time to serve one individual.
Most of the times it is convenient not to use the interarrival and departure times, but the rates (customers per time unit).

**Definition 3.1.1.** By definition $\lambda$ is the arrival rate of the queue, and $\mu$ the departure rate of the queue. Hence:

$$\lambda = \frac{1}{E[A]},$$  \hspace{1cm} (3.1.1)

and:

$$\mu = \frac{1}{E[B]}.$$ \hspace{1cm} (3.1.2)

Now it is possible to introduce the polling model.

### 3.1.2 Polling Model

A polling model is a system consisting of multiple queues, served by a single server. The server can only serve one queue at the same time. Only when a queue is being served, customers can depart. Arrivals can always take place. When a server switches to another queue, this may take some time. This time is denoted by the random variable $S$, called the switch-over time. In this report, the number of queues in the polling model is denoted by $N$.

A server in a (cyclic) polling model uses the exhaustive discipline if and only if it serves all queues one by one (first queue 1, then queue 2, and so on) and every queue is being served until it is empty.

If another discipline is being used, the presented description of a polling model might not be correct anymore. For example, suppose that the server serves a random amount of customers in every queue, then also the random variable defining this behaviour influences the state. For this report, it is assumed that all servers use the exhaustive discipline (see Section 3.2).

Note that $\mu$ and $\lambda$ might differ for every queue (and they do in general). According to Definition 3.1.1, $\mu$ is equal to the reciprocal of the expected service time. This is not equal to the departure rate anymore if that queue is placed in a polling model. That is why $\tau$ is defined. Because these parameters are unique for every queue, $\lambda_i$, $\mu_i$, and $\tau_i$ is denoted for queue $i \in \{1, 2, \cdots, N\}$.

**Definition 3.1.2.** $\tau_i$ is the intensity of the output process of queue $i \in \{1, 2, \cdots, N\}$. If $D_i(t)$ is the number of customers that have been served at queue $i$ after time $t$, then

$$\tau_i = \lim_{\alpha \to \infty} \frac{1}{\alpha} \int_0^\alpha D_i(t) \, dt.$$ \hspace{1cm} (3.1.3)

A direct consequence of this, is the following.

**Corollary 3.1.1.** For a queue that is being served, and never left, it holds that $\tau = \min(\mu, \lambda)$. If a queue is never served (i.e. the switch-over time is infinity), then $\tau = 0$. Hence, for every queue:

$$0 \leq \tau \leq \min(\mu, \lambda).$$ \hspace{1cm} (3.1.4)

### 3.1.3 Networks of polling models

Now it is possible to introduce the main concept of this report: a network of polling models.

A network of polling models is a system that consists of multiple polling models. Each polling model has its own server that serves the queues of the polling model. Also, a routing function is defined that maps a queue to a queue in a different polling model or an empty value. If customers
have been served at a queue, the routing function determines where the customer goes to. This happens by applying the function on the queue the customer just has left. If the value exists (i.e. is not the empty value), then the customer immediately travels to that queue and starts waiting there. If the value of the function is the empty value, the customer leaves the system when being served.

There is one special case. A network of two polling models where one polling model produces input for the other, is called a network of two polling models in tandem. Note that for this case, the routing function almost always returns the empty value. The only value for which the function does not return the empty value, is the queue in the first polling model that is being linked to the queue in the second polling model.

3.2 Assumptions of the network

Before one can even talk about stability, first some important assumptions will be formulated. These assumptions are also stated in [3, 4, 9, 7, 8].

- The interarrival times of the queues in the polling model are independent. This means that the random variables that describe the interarrival times ($\{A_1, A_2, \cdots, A_N\}$) are independent of each other.

- The service times of the queues in the polling model are independent of each other and of the interarrival times. This means that the random variables that describe the interarrival and the service times ($\{A_1, A_2, \cdots, A_N, B_1, B_2, \cdots, B_N\}$) are independent.

- All random variables in the travel matrix are independent of each other. They are also independent of $\{A_1, A_2, \cdots, A_N\}$ (hence of all the other random variables in the system).

- $\mu_i$ and $\lambda_i$ are finite and non-zero for all queues.

- Only the exhaustive service discipline is studied.

- Only networks without loops are considered. Why this assumptions is necessary will be discussed in the results.

3.3 Stability

Although most papers use the term stability, the definitions can differ from each other slightly because of the differences in the used models. In this report, the most convenient definition will be used.

**Definition 3.3.1** (Stability). A polling model is called stable if there exists a state that the polling model always returns to. This means that there exists a time $T < \infty$ such that if the polling model is in that state, the polling model is again in that state before $T$ time units have passed. This is also called positive recurrent.

An unstable polling model is also called saturated (a stable one unsaturated). A direct result from this, is the following (found in [4]):

**Theorem 3.3.1.** A polling model with $N$ queues and an exhaustive service is stable if, and only if

$$\rho = \sum_{i=1}^{N} \rho_i < 1,$$

(3.3.1)
with

\[ \rho_i = \lambda_i E[B_i] = \frac{\lambda_i}{\mu_i}. \quad (3.3.2) \]

Here, \( \rho_i \) is called the work load of queue \( i \) and \( \rho \) is called the total work load of the polling model.

Proof. See [9]. \qed

### 3.4 Fluid models

A fluid model is a model that replaces the individual customers by infinitely small fluid particles. Instead of individual customers, a fluid flows through a queue with intensity \( \rho_i \) and leaves the queue with intensity 1. The difference between a fluid model and a stochastic model is that a fluid model is deterministic. Using a fluid model, insight in the stability of the corresponding stochastic model can be achieved.

The polling model can be analysed as a fluid model if it is unstable. In that case, such a large number of customers is being served by one queue that the individual differences in the arrival times can not be distinguished anymore (see Figure 3.1 for an example).

A fluid model is basically applying the law of large numbers. There are two versions of the law (see Theorem 5.2.1 for the strong version), but both state that if \( X_1, X_2, \ldots \) are independent and identically distributed random variables with expected value \( \mu \), and if

\[ \bar{X}_n = \frac{1}{n}(X_1 + \ldots + X_n), \quad (3.4.1) \]

then

\[ \bar{X}_n \to \mu \text{ for } n \to \infty. \quad (3.4.2) \]

For example, suppose that a queue receives cars with exponentially distributed interarrival times with an average of one car per second. Now, when a large number of cars has arrived, the probability that the average interarrival time of the cars is \( \mu \) is pretty high.

A fluid analysis is based on this principle. If a lot of cars arrive at an intersection, this might be seen as a lot of cars arriving with a constant interarrival time (see Figure 3.1). This constant interarrival time is the expected value of the random variable that determines the interarrival time. With this type of analysis, it is much easier to prove stability.
Figure 3.1: A sample path of a single simulation run of a stable queue in an unstable polling model. As one can see, the lines are almost straight and can just be assumed to have a linear coefficient of $\mu$. 

Chapter 4

Theoretical Analysis

The theoretical analysis consists of an analysis of the described model. For this analysis, a fluid model will be used to determine whether a polling model is stable or not. Using a fluid model, one can keep track of the several queue lengths after each queue has been served.

The first step is to analyse what stability of a network means (as this is not defined yet) and analyse a simple example. Then, these results can be extended to a more general case.

All the switch-over times are assumed to be zero. This can be done because a fluid analysis is not influenced by switch-over times (see Figure 3.1).

4.1 Stable situation

A polling model is saturated if and only if \( \sum_{i=1}^{N} \rho_i < 1 \) (see Theorem 3.3.1), with \( \rho_1, \rho_2, \ldots, \rho_N \) the work loads of the queues. It is possible to extend Corollary 3.1.1 for the situation where both polling models are unsaturated.

**Theorem 4.1.1.** If a polling model is stable, then \( \tau_i = \min(\mu_i, \lambda_i) = \lambda_i \) for each queue \( i \).

**Proof.** If the polling model is stable, the individual queues are also stable. Hence for the specific queue \( i \):

\[
\rho_i = \frac{\lambda_i}{\mu_i} < 1,
\]

thus:

\[
\lambda_i < \mu_i.
\]

(4.1.2)

So the following is trivial:

\[
\lambda_i = \min(\mu_i, \lambda_i).
\]

(4.1.3)

From Corollary 3.1.1, it follows that:

\[
\tau_i \leq \min(\mu_i, \lambda_i).
\]

(4.1.4)

To finalize the proof, it is necessary to prove that \( \tau_i \geq \min(\mu_i, \lambda_i) \). This will be proved using a proof by contradiction. Assume that \( \tau_i < \min(\mu_i, \lambda_i) \). Then \( \tau_i < \lambda_i \). This is not possible because the polling model is stable (a recurrent state would not exist).
4.2 One unstable polling model

Now assume that the one polling model is saturated, exists of two queues (with $\rho_{1,1}$ and $\rho_{1,2}$) and produces output for a second polling model with two queues ($\rho_{2,1}$ and $\rho_{2,2}$). Hence:

$$\rho_{1,1} < 1 \text{ and } \rho_{1,2} < 1 \text{ but:}$$

$$\rho_1 = \rho_{1,1} + \rho_{1,2} > 1 \text{ and:}$$

$$\rho_2 = \rho_{2,1} + \rho_{2,2} < 1$$

Note that if queue 1 of the second polling model receives the input from the first polling model, that:

$$\rho_{2,1} = \frac{\tau_{1,2}}{\mu_{2,1}}$$

(4.2.1)

The question is what will happen with the second polling model, and when will that polling model be unstable? To answer this question, one wants to calculate the intensity $\tau_{1,2}$ of the departure process of the first polling model (the unstable one). Then the question can be answered what will happen with the second polling model.

For this calculation, a fluid model can be used. With a fluid model, it is possible to calculate the fluid level (work at a queue) in each queue at visit beginnings.

The model starts with an empty system, except for one queue (which without loss of generality is assumed to be queue 1). By assuming that queue 1 is non-empty, the assumption is made that at time $t = 0$ the server starts serving this queue with $\epsilon$ work. With this assumption, a fluid analysis can be made (Table 4.1). Without this assumption, the fluid model depends on the queue that receives work first.

Some remarks about this analysis can be stated:

1. For this special case, the stability condition from Theorem 3.3.1 can be proved from this analysis.

   *Another proof of Theorem 3.3.1 for two queues.* As can be seen in the last two rows in the table, the amount of work is multiplied with the factor

   $$\frac{\rho_1 \rho_2}{(1 - \rho_1)(1 - \rho_2)}.$$  

(4.2.2)

So the (expected) amount of work in each queue does not grow if and only if this factor is less than one. This is also a necessary and sufficient condition for stability. Hence:

$$\frac{\rho_1 \rho_2}{(1 - \rho_1)(1 - \rho_2)} < 1 \Rightarrow \rho_1 \rho_2 < (1 - \rho_1)(1 - \rho_2)$$

$$\Rightarrow \rho_1 \rho_2 < 1 - \rho_1 - \rho_2 \Rightarrow 0 < 1 - \rho_1 - \rho_2 \Rightarrow \rho_1 + \rho_2 < 1.$$

2. The maxima of the total amount of work in each queue decrease or increase linearly over time. The linear coefficient can be calculated. For this, take the first two points

$$(0, \epsilon) \rightarrow (\frac{1}{1 - \rho_1} + \epsilon \frac{\rho_2}{(1 - \rho_1)(1 - \rho_2)}, \epsilon \frac{\rho_1 \rho_2}{(1 - \rho_1)(1 - \rho_2)})$$
<table>
<thead>
<tr>
<th>Time</th>
<th>Queue 1</th>
<th>Queue 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\epsilon$</td>
<td>0</td>
</tr>
<tr>
<td>$\epsilon \frac{1}{1 - \rho_1}$</td>
<td>0</td>
<td>$\epsilon \frac{\rho_2}{1 - \rho_1}$</td>
</tr>
<tr>
<td>$\epsilon \frac{\rho_2}{(1 - \rho_1)(1 - \rho_2)}$</td>
<td>$\epsilon \frac{\rho_1 \rho_2}{(1 - \rho_1)(1 - \rho_2)}$</td>
<td>0</td>
</tr>
<tr>
<td>$\epsilon \frac{\rho_1 \rho_2}{(1 - \rho_1)^2(1 - \rho_2)}$</td>
<td>0</td>
<td>$\epsilon \frac{\rho_1 \rho_2^2}{(1 - \rho_1)^2(1 - \rho_2)}$</td>
</tr>
<tr>
<td>$\epsilon \frac{\rho_1 \rho_2^2}{(1 - \rho_1)^3(1 - \rho_2)^2}$</td>
<td>0</td>
<td>$\epsilon \frac{\rho_1 \rho_2^3}{(1 - \rho_1)^3(1 - \rho_2)^2}$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$\epsilon \frac{\rho_1^{n-1} \rho_2^n}{(1 - \rho_1)^n (1 - \rho_2)^n}$</td>
<td>$\epsilon \frac{\rho_1^n \rho_2^n}{(1 - \rho_1)^n (1 - \rho_2)^n}$</td>
<td>0</td>
</tr>
<tr>
<td>$\epsilon \frac{\rho_1^n \rho_2^n}{(1 - \rho_1)^n+1 (1 - \rho_2)^n}$</td>
<td>0</td>
<td>$\epsilon \frac{\rho_1^n \rho_2^{n+1}}{(1 - \rho_1)^n+1 (1 - \rho_2)^n}$</td>
</tr>
</tbody>
</table>

Table 4.1: The amount of fluid in the fluid model after each queue has been served. When queue 1 has been served having $w$ work, this takes $w/(1 - \rho_1)$ time. In the mean time, queue 2 receives $\rho_2 w/(1 - \rho_1)$ work. This iteration is repeated for every queue, and a general equation can be derived after $n$ iterations.
for the first queue and
\[
(\epsilon \frac{1}{1 - \rho_1}, \epsilon \frac{\rho_2}{1 - \rho_1}) \rightarrow \\
(\epsilon \frac{1}{1 - \rho_1} + \epsilon \frac{\rho_2}{(1 - \rho_1)(1 - \rho_2)} + \epsilon \frac{\rho_1\rho_2}{(1 - \rho_1)^2(1 - \rho_2)}, \epsilon \frac{\rho_1\rho_2^2}{(1 - \rho_1)^2(1 - \rho_2)})
\]
for the second, and calculate the coefficient. Then for the first queue, one gets:
\[
\frac{\epsilon \frac{\rho_1\rho_2}{(1 - \rho_1)(1 - \rho_2)} - \epsilon}{\frac{1}{1 - \rho_1} + \epsilon \frac{\rho_2}{(1 - \rho_1)(1 - \rho_2)} - 0} = \\
\frac{\frac{1}{1 - \rho_1} + \epsilon \frac{\rho_2}{(1 - \rho_1)(1 - \rho_2)} - 1}{\frac{\rho_1\rho_2}{(1 - \rho_1)(1 - \rho_2)}} = \\
\frac{\rho_1\rho_2 - (1 - \rho_1)(1 - \rho_2)}{(1 - \rho_2) + \rho_2} = \\
\rho_1\rho_2 - (1 - \rho_2) - 1,
\]
The second queue yields the same result.

3. The coefficient (see previous point) of the linear growth of the work in the queues is 
\[
\rho_1 + \rho_2 - 1
\]
which is positive if the polling model is unstable. So if the model is unstable, 
the work that has to be served in the system keeps growing.

4. The linear coefficient is the amount of fluid that enters the polling model \((\rho_1 + \rho_2)\), with 
the amount that departs subtracted \((1)\).

4.3 \(N\) queues

It is possible to extend this result to \(N \geq 2\) queues. Assume that there exists a polling model 
with \(N\) queues in a network. If the departure process of this polling model can be determined, 
then it is again possible to determine if the polling models in the network are stable. This might 
become a puzzle (because they all influence each other), but if the network does not have any 
cycles, then this puzzle is easy to solve. Note that if the network has any cycles, the arrival 
intensities of the queues become dependent of each other. For this, the set up must be extended 
and vectors must be introduced. Then, a function can be defined to serve one queue.

4.3.1 The situation

Suppose there is a polling model with \(N\) queues. Each queue has an arrival intensity of \(\lambda_i\), 
service time of \(B_i\) and hence a work load of \(\rho_i\). For the polling model, it is convenient to use 
these factors in vector format. So denote \(\vec{R} = (\rho_1, \rho_2, \ldots, \rho_N)^T\), and \(\vec{R}_k = (\rho_1, \rho_2, \ldots, \rho_{k-1}, 0, \rho_{k+1}, \ldots, \rho_N)^T\). There are no switch-over times.

For the same reason as with two queues, there must be some work in the system when the 
calculations start. This work, denoted by \(\vec{w}\) (a vector of length \(N\)), will not be fixed this time 
but can be chosen arbitrarily. Now, a function \(\Omega_s : \mathbb{R}^N \rightarrow \mathbb{R}^N\) that transforms the vector \(\vec{w}\) into 
a new \(\vec{w}^*\) that is the situation of the work in the polling model after a queue \(s\) has been served 
can be determined. Then, this function can be applied for every \(s\). This can be repeated many 
times to find out the rate at which work is floating out the polling model.
4.3.2 Serving one queue

Suppose \( \vec{w} = (w_1, w_2, \ldots, w_N) \) as above, so \( w_1 \) is the amount of work at queue 1, \( w_2 \) the amount of work at queue 2 and so on. Then suppose the server switches to queue \( s \neq 1 \) and starts serving all the customers exhaustively. Again, a fluid analysis can be made. Then \( \vec{w}' = \Omega_s(\vec{w}) \) denotes the work after the server is done. It is easy to determine the form of this vector. If queue \( s \) is served, then \( w_1' = w_1 + \frac{w_s}{1 - \rho_s} \rho_1 \), because the work that was already there stays there and it takes \( \frac{w_s}{1 - \rho_s} \) time to serve queue \( s \), so that amount of work is added. This equation is also true for all other queues, except of course for queue \( s \). That queue is empty after being served. Hence:

\[
\Omega_s(\vec{w}) = (w_1 + \frac{w_s}{1 - \rho_s} \rho_1, w_2 + \frac{w_s}{1 - \rho_s} \rho_2, \ldots, w_{k-1} + \frac{w_s}{1 - \rho_s} \rho_{k-1}, 0, w_{k+1} + \frac{w_s}{1 - \rho_s} \rho_{k+1}, \ldots, w_N + \frac{w_s}{1 - \rho_s} \rho_N).
\]

(4.3.1)

Now a theorem about a property of this function can be formulated.

**Theorem 4.3.1.** The function \( \Omega_s \) is linear for all \( s \).

**Proof.** The function \( \Omega_s \) is linear if and only if \( \Omega_s(\alpha \vec{w} + \beta \vec{v}) = \alpha \Omega_s(\vec{w}) + \beta \Omega_s(\vec{v}) \) for all \( \vec{v}, \vec{w} \in \mathbb{R}^N \), \( \alpha, \beta \in \mathbb{R} \). This can be proved:

\[
\Omega_s(\alpha \vec{w} + \beta \vec{v}) = \\
(\alpha w_1 + \frac{\alpha w_s + \beta v_s}{1 - \rho_s} \rho_1, \ldots, 0, \ldots, \alpha w_N + \frac{\alpha w_s + \beta v_s}{1 - \rho_s} \rho_N) = \\
(\alpha w_1 + \frac{\alpha w_s + \beta v_s}{1 - \rho_s} \rho_1, \ldots, 0, \ldots, \alpha w_N + \frac{\alpha w_s + \beta v_s}{1 - \rho_s} \rho_N) + \\
(\beta v_1 + \frac{\beta v_s - \rho_1}{1 - \rho_s}, \ldots, 0, \ldots, \beta v_N + \frac{\beta v_s - \rho_N}{1 - \rho_s}) = \\
\alpha(\vec{w}_1 + \frac{w_s}{1 - \rho_s} \rho_1, \ldots, 0, \ldots, w_N + \frac{w_s}{1 - \rho_s} \rho_N) + \\
\beta(\vec{v}_1 + \frac{v_s}{1 - \rho_s} \rho_1, \ldots, 0, \ldots, v_N + \frac{v_s}{1 - \rho_s} \rho_N) = \\
\alpha \Omega_s(\vec{w}) + \beta \Omega_s(\vec{v}).
\]

\( \square \)

**Corollary 4.3.1.** There exists a \( N \times N \)-matrix \( O_s \) such that \( \Omega_s(\vec{v}) = O_s \vec{v} \) for all \( \vec{v} \in \mathbb{R}^N \).

The question is how this matrix can be written. This question is not that hard to answer.
4.3. N QUEUES

CHAPTER 4. THEORETICAL ANALYSIS

Theorem 4.3.2. A \( N \times N \)-matrix \( O_s \) of the function \( \Omega_s \) is equal to:

\[
O_s = \begin{pmatrix}
1 & 2 & 3 & s-1 & s & s+1 & N \\
1 & 0 & 0 & \cdots & 0 & \frac{\rho_1}{1-\rho_s} & 0 & \cdots & 0 \\
2 & 0 & 1 & 0 & \cdots & 0 & \frac{\rho_2}{1-\rho_s} & 0 & \cdots & 0 \\
3 & 0 & 0 & 1 & \cdots & 0 & \frac{\rho_3}{1-\rho_s} & 0 & \cdots & 0 \\
& \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
N & 0 & 0 & 0 & \cdots & 0 & \frac{\rho_N}{1-\rho_s} & 0 & \cdots & 1 \\
\end{pmatrix}
\] (4.3.2)

Proof. The columns of the matrix \( O_s \) can be determined by evaluating \( \Omega_s \) with the base vectors of \( \mathbb{R}^N \). Those base vectors (the columns of the identity matrix \( I \)) consist of \( \vec{e}_1 = (1, 0, \ldots, 0)^T \), \( \vec{e}_2 = (0, 1, 0, \ldots, 0)^T \) and so on. Hence for \( s > 3 \):

\[
\Omega_s(\vec{e}_1) = (1, 0, 0, \ldots, 0)^T \\
\Omega_s(\vec{e}_2) = (0, 1, 0, \ldots, 0)^T \\
\Omega_s(\vec{e}_3) = (0, 0, 1, \ldots, 0)^T,
\]
because queue \( s \) is being served. Queue \( s \) is empty, so nothing happens. The same holds for every other \( e_i \) with \( i \neq s \). When there is only work at queue \( s \) and not at the other queues, this does not work anymore. The other queues then receive work when queue \( s \) is being served. So:

\[
\Omega_s(\vec{e}_s) = \left( \frac{\rho_1}{1-\rho_s}, \frac{\rho_2}{1-\rho_s}, \ldots, \frac{\rho_N}{1-\rho_s} \right).
\] (4.3.3)

\[\square\]

4.3.3 Serving a cycle

It is easy to extend the result of serving one queue to serving all \( N \) queues. This can be done by using function composition. If at the beginning of the cycle there is \( \vec{w} \) amount of work in the polling model, then after serving \( N \) queues starting at queue 1, one gets as work \( \vec{w}^* \):

\[
\vec{w}^* = (\Omega_N \circ \Omega_{N-1} \circ \cdots \circ \Omega_2 \circ \Omega_1) \vec{w}.
\] (4.3.4)

Hence:

\[
\vec{w}^* = (O_N O_{N-1} \cdots O_2 O_1) \vec{w}.
\] (4.3.5)

Although it is possible to write this matrix on a piece of paper, its elements would be too complex and too big to write down. So that is why this report will not contain this matrix. More about this in chapter 5.
Chapter 5

Implementation

Because the matrix in Equation (4.3.5) is too big to write out, a computer was used to do this numerically. First, the model was implemented in the program “Mathematica” and with this program it was possible to determine the results. Secondly, a (stochastic) simulation was written to ensure that the results were correct. This chapter describes both situations in pseudo-code, results can be found in the next chapter.

5.1 Numerical implementation

Actually, implementing the model is pretty straight-forward. First, one generates the $N \times N$-matrix $O_s$ for every $s$ (Algorithm 5.1).

\begin{algorithm}
\caption{Generating the $O_s$-matrix}
\begin{algorithmic}[1]
\Require $N, \rho_1, \ldots, \rho_N$
\For{$s = 1 \rightarrow N$} \Comment Construct the $O_i$ matrix for every $i$
    \For{$i = 1 \rightarrow N$}
        \For{$j = 1 \rightarrow N$}
            \If{$i = s \land j = s$}
                $O_s[i][j] = 0$
            \ElsIf{$i = j$}
                $O_s[i][j] = 1$
            \ElsIf{$j = s$}
                $O_s[i][j] = \rho_i/(1 - \rho_s)$
            \Else
                $O_s[i][j] = 0$
            \EndIf
        \EndFor
    \EndFor
\EndFor
\end{algorithmic}
\end{algorithm}

Then, given a work vector, the problem that exists is multiplying matrices (Algorithm 5.2). This algorithm uses $\text{Bucket}$ to store the work that has left a queue. $\text{Bucket}_i$ consists of the amount of work that has been served by queue $i$.

When the algorithm has run, the vector $\text{Bucket}$ is the state of the system after $M$ cycles.
Algorithm 5.2 Iterating and simulating

Require: \( N, \rho_1, \ldots, \rho_N, O_1, \ldots, O_N, M, \vec{w} \)  \( \triangleright M \) is the amount of iterations

\[ \text{Time} \leftarrow 0 \]
\[ \text{Bucket} \leftarrow \{0\}^N \] \( \triangleright \) A vector with \( N \) zero’s

\begin{algorithmic}
   \For {\( i = 1 \rightarrow M \)}
      \For {\( j = 1 \rightarrow N \)}
         \State \( \text{Time} \leftarrow \text{Time} + \frac{w_i}{1 - \rho_i} \)
         \State \( \text{Bucket}_i \leftarrow \text{Bucket}_i + \frac{w_i}{1 - \rho_i} \)
         \State \( \vec{w} \leftarrow O_i \vec{w} \)
      \EndFor
   \EndFor
\end{algorithmic}

Now, the most convenient way to calculate \( \tau_i \) would be to divide the amount of customers that have left (in \( \text{Bucket} \), note that this is denoted by \( D(t) \)) with the time. Although this value is related with \( \tau_i \), they are not equal (see Figure 5.1 for an image of this value). The problem is that \( \tau \) is not related with \( D(t) \), but with \( \mathbb{E}[D(t)] \):

\[
\tau = \lim_{\alpha \to \infty} \int_{0}^{\alpha} \frac{D(t)}{\alpha} \, dt \quad (5.1.1)
\]

As one can see in the picture, the factor varies over time. This can be explained. Suppose in the picture is a plot of a function \( T \). Suppose \( a \) is the time just after the queue has been served, \( b \) is the time just before the next serving of the queue and \( c \) is the time when the queue is served again. Note the following:

- \( T(a) = \lambda \), because the queue is completely empty when every customer has been served, so the arrival intensity is the depart intensity.
- \( T(b) \) is a local minimum of the function. Also, as the time increases, it does not matter anymore where the measuring starts.
- \( T(c) = \lambda \), because the queue is empty again.
- From \( T(b) \) to \( T(c) \) is the function \( T \) linear with coefficient \( 1/(1 - \rho) \), where \( \rho \) is the work load of the queue.
- From \( T(a) \) to \( T(b) \), the amount of work in the bucket stays constant, hence the \( T(t) = \frac{aT(a)}{t} \) for \( a < t < b \).

Now a good estimation of \( \tau \) can be made for \( t \) big. Hence for \( a, b, c \) large:

\[
\tau = \int_{a}^{b} T(a) \, dx + \frac{T(b)+T(a)}{2} \frac{c-a}{c-a}
= aT(a) \log \left( \frac{b}{a} \right) + \frac{T(b)+T(a)}{2} \quad (5.2)
\]

5.2 Simulation

To verify the results found so far, a simulation was written of the stochastic model. Of course, a simulation on its own cannot prove any results, but when the results from a simulation are the same as the results from the theory, the chance that the theory is incorrect might be reduced.
Figure 5.1: The plot of the amount of customers that have left a queue divided with the time so far \( \frac{D(t)}{t} \) for one simulated run. When a server is serving the queue, this factor increases to \( \lambda \). If a server is not serving the queue, this factor decreases hyperbolic. The value of \( \lambda \) is 0.1 and \( \tau > \lambda \) can happen because this is the result of a stochastic simulation.
But before one can talk about a simulation, some simulation concepts must be introduced. As a simulation is just a computer program, one can simulate far more complex situations without any assumptions. This is why only queues and networks are being used in this chapter (no fluid models anymore).

### 5.2.1 Monte-Carlo simulations

At the beginning of the second World War, the first computer was created. Although this computer could not be compared to the personal computers we know today, it was able to calculate (simple) expressions. One problem arose: a computer was deterministic, it could not generate a random number. In the beginning of the evolution of the computer this was solved by just entering some numbers in the computer and using those number as random number, but this was not a perfect solution. After a while, someone invented the pseudo random number (by using a linear congruential generator, see [2] for a visualisation), and the first simulation was invented.

In most of the cases, a computer is only able to draw a uniform distributed random variable in the range $(0, 1)$. Luckily some (fast!) methods exist to transform such a number to a random variable with another distribution. Suppose that $U \in (0, 1)$ is the generated uniform random variable.

#### Inverse Transform Method

Let the random variable $X$ have a continuous and increasing distribution function $F$. Denote the inverse of $F$ by $F^{-1}$. Then $X$ can be generated as follows:

$$X = F^{-1}(U),$$

or, if $F$ is not continuous:

$$X = \min\{x : F(x) \geq U\}.$$  

(5.2.1)

(5.2.2)

For example, now $X = -\ln(1 - U)/\lambda$ is an exponentially distributed random variable with parameter $\lambda$.

Of course, this method does not always apply, as most distribution functions do not have an easy-to-represent inverse. If the function can be written as a sum of other distributions, some tricks can be applied, but this is not always the case. For this, the acceptance-rejection method is a solution.

#### Acceptance-Rejection method

If $X$ has probability density function $f$, and $g \geq f \forall x$, then $h(x) = g(x)/c$ is a density if $c = \int_{-\infty}^{\infty} g(x) \, dx$. With the following algorithm, $X$ can be generated:

- Generate $Y$ having density $h$;
- Generate $U$ independent of $Y$;
- If $U \leq f(Y)/g(Y)$, then $X = Y$, else start over again.

It is not trivial that now $X$ has the correct distribution, although this can be proved.

*Proof.* Because of the last step, the following is true:

$$P(X \leq x) = P(Y \leq x \mid Y \text{ accepted}).$$  

(5.2.3)
Now, the right side of this equation can be calculated by writing out the conditional probability:

$$P(Y \leq x \mid Y \text{ accepted}) = \frac{P(Y \leq x \text{ and } Y \text{ accepted})}{P(Y \text{ accepted})}.$$ (5.2.4)

Because

$$P(Y \leq x \text{ and } Y \text{ accepted}) = \int_{-\infty}^{x} f(y) h(y) \, dy = \frac{1}{c} \int_{-\infty}^{x} f(y) \, dy,$$ (5.2.5)

and by letting $x \to \infty$

$$P(Y \text{ accepted}) = \frac{1}{c},$$ (5.2.6)

we have:

$$P(X \leq x) = \int_{-\infty}^{x} f(y) \, dy.$$ (5.2.7)

Although faster methods exist in practice, this method does not require strong assumptions.

**Monte-Carlo**

Most probabilistic simulations use the Monte-Carlo principle (see [1, 6]). The Monte-Carlo principle is based on the principle of the **Law of large numbers**:

**Theorem 5.2.1** (Strong law of Large numbers). If $\bar{X}_n = \frac{1}{n} \sum_{i=1}^{n} X_i$, then

$$P\left(\lim_{n \to \infty} \bar{X}_n = \mu\right) = 1.$$ (5.2.8)

So why is this law necessary? The problem is that it is only possible to simulate one specific instance of the problem at once. Most of the times, it is not interesting to know what happens in one specific situation, but someone would like to know what happens on average. So by this law, it is possible to just simulate everything many times and then take the mean of the results.

**5.2.2 Discrete-event simulations**

The last question that arises is: how to simulate just one situation? There are several possibilities, but for a queuing (non-continuous) situation, the discrete-event simulation is the most convenient.

A discrete-event simulation uses a priority queue (see [5] for more information about the implementation of this structure) to save events. Every event has a time property that consists of the exact time the event (should) happen. The priority of the event is its time property, and the element with the lowest property should be returned first (hence the lowest time, hence first to happen).

When the simulation starts, the first event or events that happen are added to the priority queue. Then, the program starts an infinite loop (that is stopped after a period has elapsed or the program has simulated a certain amount of time) doing the following:

1. Pop the first event from the priority queue, this is the event with the lowest time in the queue.
2. Process the event and change the state of all the thing according to the data of the event.
3. Now, it is also the time to create new events and add them to the priority queue.
4. Destroy the event that was being processed (from step 1) and start over again.

A discrete event simulation is one of the fastest methods to simulate situations where events happen consecutively in discrete time laps.
5.2.3 A simple example

Now put all the descriptions together in a simple example. Suppose $Q$ is an M/M/1-queue. To create a program that can simulate the behaviour of this queue, the first problem is to get a random variable with an exponential distribution. For this, the inverse transform method can be used.

It is also necessary to describe how events work in the simulation (see definition 5.2.1).

**Definition 5.2.1.** An event of the described simulation consists of a status code and a time. The time is the time at which the event happens, and the status code describes what happens. If the status code is 0, then a customer arrives at the queue. If the status code is 1, then a customer has been served and leaves the queue.

With this information, the simulation can be written. An implementation of a min-priority queue is needed as event queue. For this, a min-heap can be used (more information can be found in [5]).

**Algorithm 5.3** Simulating a simple $M/M/1$-queue

**Require:** $pq$: the event queue, $\lambda$: the arrival intensity, $\mu$: the depart intensity, $U()$: a function to draw a uniform random variable between 0 and 1, $E(u, \lambda)$: a function to draw an exponential distributed random variable with parameter $\lambda$ using a uniform distributed random number $u$.

$Q \leftarrow 0$  
$\triangleright$ $Q$ is the number of customers in the queue
Create a new event with status code 0 (arrival) and time $T + E(U(), \lambda)$ and add it to $pq$. This is the first arrival at the queue.

**while** The user has not interrupted the simulation yet **do**

Remove the first item from $pq$ and advance $T$ to the time of this event.

**if** The status code of the event is 0 **then**

$Q \leftarrow Q + 1$  
$\triangleright$ An arrival
Add a new event with status code 0 and time $T + E(U(), \lambda)$ and add it to $pq$.

**if** $Q = 1$ **then**

Add a new event with status code 1 and time $T + E(U(), \mu)$ and add it to $pq$.

**end if**

**else**

$Q \leftarrow Q - 1$  
$\triangleright$ A customer has been served

**if** $Q > 0$ **then**

Add a new event with status code 1 and time $T + E(U(), \mu)$ and add it to $pq$.

**end if**

**end if**

**end while**
Chapter 6

Main results

In this section, numerical results will be obtained from the implemented fluid model in Mathematica and from the stochastic simulation. If everything is correct, those are the same. Discussion of the results will happen in the next chapter.

6.1 Fluid model

First, the situation will be discussed that all polling models are stable, and remain stable. Then, unstable polling models will be discussed.

6.1.1 Stable situation

Suppose that there are several polling models linked together in a network, that they produce output for each other and that they are all stable even when they receive output from each other. Stability can be proved by using the presented theorem (see Theorem 3.3.1).

Now the departure rates can be determined, and are actually equal to the arrival rates (see Theorem 4.1.1).

6.1.2 Two queues

For a polling model with two queues, the calculated result of a fluid analysis was presented in Table 4.1. It is possible to obtain symbolic results from this table, but the expressions would become very complex. That is why numerical methods have been used, and the results are the same as for \( N \) queues (see next chapter).

6.1.3 \( N \) queues

For 2, or even \( N \) queues, one can use the two Algorithms 5.1 and 5.2 to determine good numerical estimators of \( \tau \). To do this, one has to take several unstable polling models and then calculate their departure rates (Table 6.1). Then, in the next section, these situations will be simulated and then it can be checked whether these results are true. For these calculations, the program Mathematica is used.

Note that the results are not that trivial. The representation of the numbers is not exact, and there is no easy-to-use expression to calculate these results. However, there might exist an
exact formula to calculate these numbers, although we have to been able to determine such an expression.

6.2 Simulation

Now take the situations of Table 6.1 and check whether a simulation of the described situations delivers the same results as the results in the table. To check this, the described simulation in the previous chapter is used. The number of processed work is counted for every queue. If a plot is made of that number divided by the time, then this delivers a good estimator of $\tau$ if the average of many runs is taken. Remember that the simulation has to run a lot of times, otherwise the individual random variables have too much influence.

The nine described simulations were simulated using a personal computer. The simulation was cut off when 100,000 time units had passed. The simulation was run 250 times, and the average of all the results was taken. Then, $\tau$ was estimated for every time unit. This means that every time unit, the processed work was taken and divided by the time. The longer one simulates, and the more one simulates, the more precise this estimation gets. The results are presented in plots in Appendix A. The vertical lines in the plots are the numbers from Table 6.1.

To check whether the amount of runs would influence the results even further, one very long simulation of situation 8 was run. Then, it can be checked whether the result differ much, and whether the amount of runs would be a point of discussion. The simulation was cut off after 2,500,000 time units and was run 2,000 times. The result of this simulation can also be found in Appendix A.

6.3 Interpretation

The results from the fluid model and the results from the stochastic simulation seem to be equal, but what does it all mean? In this section, some typical examples will be presented. Note again that the results are not that trivial. One might expect that the departure rates of the queues of a polling model have the same ratio as the arrival rates. Hence the departure rate of situation 8 would be for the first queue: $0.2 \times 0.3 + 0.4 + 0.5 = 0.14$. The same calculation can be used for the other queue, which yields $0.2, 0.29$ and $0.36$, but this is not the correct. The queues with a small arrival rate tend to have a smaller departure rate than expected. The queues with bigger arrival rates seem to have a bigger departure rate than expected.

<table>
<thead>
<tr>
<th>#</th>
<th>$R$</th>
<th>$T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{0.4, 0.7}</td>
<td>{0.348399, 0.651601}</td>
</tr>
<tr>
<td>2</td>
<td>{0.5, 0.7}</td>
<td>{0.396232, 0.603768}</td>
</tr>
<tr>
<td>3</td>
<td>{0.6, 0.7}</td>
<td>{0.446149, 0.553851}</td>
</tr>
<tr>
<td>4</td>
<td>{0.7, 0.7}</td>
<td>{0.5, 0.5}</td>
</tr>
<tr>
<td>5</td>
<td>{0.3, 0.4, 0.5}</td>
<td>{0.241576, 0.33161, 0.426814}</td>
</tr>
<tr>
<td>6</td>
<td>{0.4, 0.3, 0.5}</td>
<td>{0.33161, 0.241576, 0.426814}</td>
</tr>
<tr>
<td>7</td>
<td>{0.4, 0.4, 0.5}</td>
<td>{0.302696, 0.302696, 0.394607}</td>
</tr>
<tr>
<td>8</td>
<td>{0.2, 0.3, 0.4, 0.5}</td>
<td>{0.129861, 0.205055, 0.28745, 0.377634}</td>
</tr>
<tr>
<td>9</td>
<td>{0.1, 0.2, 0.3, 0.4, 0.5}</td>
<td>{0.0564705, 0.120414, 0.191978, 0.2715, 0.359638}</td>
</tr>
</tbody>
</table>

Table 6.1: The results of the model for several values of $\rho$. 

24
6.3. INTERPRETATION

A small example, is an example of two in tandem placed polling models (see Figure 6.1). Both polling models seem to be stable, and this is the case. To prove this, one has to start the calculation with the left polling model (that one does receive output from another polling model). That polling model is stable, hence the input on the right polling model becomes 0.3. And then the right polling model is stable, and the output on the road to the right becomes also 0.3.

Now the next example is discussed (see Figure 6.2). The only difference between the two examples is that the first one consists of two *stable* polling models, and the second does not. Now if the arrival rate of 0.4 would also become the output intensity, the second polling model would become unstable. But this is not the case. If the numbers are plugged in the fluid model, the depart rates become like the simulated situation 5, hence the input on the second polling model would become 0.33. This is small enough to make the second polling model stable (the sum is 0.98).

![Figure 6.1: A small network without loops. Here, it is assumed that all the service times are equal to 1 (so $\rho = \lambda$ for each queue).](image-url)
Figure 6.2: A small network without loops with one unstable polling model. Here, it is assumed that all the service times are equal to 1 (so $\rho = \lambda$ for each queue).
Chapter 7

Conclusion

The described results yield immediately an answer to the question when a customer has a finite processing time in a system of polling models. Note that the network cannot have any loops.

- If all the polling models are stable (even when linked together), then there is an upper limit of the total time it takes to travel through the network. This upper limit does not depend on the queue where the customer starts.

- If all the polling models are unstable, then there is no upper limit. If the customer enters the system (the queue does not matter), one cannot predict how long it will take until the customer has left the system. Of course, one can use more information, like the customers in the system, but that is not meant here.

- If some polling models are unstable, the output of those polling models can be calculated using the described method. With these outputs it is possible to determine whether there are more unstable polling models in the network.

- If a polling model is unstable, and a customer that is travelling through the network and uses the intersection that that polling model represents, there does not exist an upper limit. This is because the queue lengths can have any arbitrary size.

Also, a simulation can be used to check whether the calculated results are true, although the situations are unstable and require a lot of memory.

7.1 Further research

Although the several aspects were discussed with as much detail as possible, there were several aspects that could be extended in a further research. One aspect is that every calculation, simulation and model uses the expected values of the random variable. Distributions of random variables were not mentioned. The question might arise if it is even possible to do this (due to the instable situation, the distribution might have an infinite second moment), but this was not studied.

A second aspect that might need some clarification was the independency of the arrival probabilities of the queues in the polling model. Due to this, only a conclusion could be drawn about a network of polling models without loops.

Also, several researchers (see, for example, [4]) already simulated some polling models and derived approximations of several distributions. One might question whether these approximations
still work in a network of polling models, and if they work, what the error of the approximation is.
Bibliography


Appendices
Appendix A

Plots of the results

In Chapter 6 is told about the results of a simulation. In this appendix, all the pictures of the simulations are presented.
Figure A.1: Situation 1
Figure A.2: Situation 2
Figure A.3: Situation 3
Figure A.4: Situation 4
Figure A.5: Situation 5
Figure A.6: Situation 6
Figure A.7: Situation 7
Figure A.8: Situation 8
Figure A.9: Situation 8, simulated for 2,500,000 time units and 2,500 times.
Figure A.10: Situation 9