Decomposition-Based Analysis of Queueing Networks

Ramin Sadre

CTIT Ph.D.-thesis Series No. 06-95
Centre for Telematics and Information Technology
University of Twente, P.O. Box 217, NL-7500 AE Enschede
ISSN 1381-3617

ISBN 90-365-2444-X

DECOMPOSITION-BASED ANALYSIS OF QUEUEING NETWORKS

DISSERTATION

to obtain
the degree of doctor at the University of Twente,
on the authority of the rector magnificus,
prof. dr. W.H.M. Zijm,
on account of the decision of the graduation committee,
to be publicly defended
on Wednesday, January 10th, 2007 at 13.15

by

Ramin Sadre

born on June 13th, 1974
in Tehran (Iran)

This dissertation has been approved by
the promotor, prof. dr. ir. B.R. Haverkort.

Abstract

Model-based numerical analysis is an important branch of the model-based performance evaluation. Especially state-oriented formalisms and methods based on Markovian processes, like stochastic Petri nets and Markov chains, have been successfully adopted because they are mathematically well understood and allow the intuitive modeling of many processes of the real world. However, these methods are sensitive to the well-known phenomenon called state space explosion. One way to handle this problem is the *decomposition approach*.

In this thesis, we present a decomposition framework for the analysis of a fairly general class of open and closed queueing networks. The decomposition is done at queueing station level, i.e., the queueing stations are independently analyzed. During the analysis, traffic descriptors are exchanged between the stations, representing the streams of jobs flowing between them. Networks with feedback are analyzed using a fixed-point iteration.

Based on the decomposition framework, we have developed an efficient analysis method called *FiFiQueues*. The method supports open queueing networks with infinite and finite capacity queues. We present the method and discuss its fixed-point behavior, as well as various extensions to the original algorithm. We also show how the method can be applied in the efficient analysis of closed queueing networks.

The service processes in FiFiQueues can be arbitrary phase-type renewal processes. Over the last decade, traffic measurements have shown the presence of properties such as self-similarity and long-range dependency in network traffic. It has been shown that this can be explained by the heavy-tailedness of many of the involved distributions. We describe how hyper-exponential distributions, which are a special case of phase-type distributions, can be fitted to heavy-tail distributed measurement data.

FiFiQueues' traffic descriptors are based on the first and second moment of the inter-arrival time and, hence, cannot account for correlations in the traffic streams. To approach this problem, we also introduce MAPs as traffic descriptors. Since a queueing network analysis based on MAP traffic descriptors suffers under the state space explosion problem, we present five different MAP reduction methods in order to decrease the effect of the state space explosion.

Contents

Chapter 1

Introduction

From the beginning of modern telecommunication systems, their performance evaluation has been of particular interest. The complexity and costs of the involved technologies and the rapidly increasing demand made it necessary to carefully plan communication networks from a technological and an economical point of view. As a consequence, the question regarding the optimal amount of resources to fulfill given or future demands arose. Basically, performance evaluation of a system should give an answer to this question.

Of course, the evaluation has to follow the same economical considerations as the evaluated object. Hence, the most direct approach, the *measurement*, is usually not an option since it requires an installed and running system. *Model-based* approaches try to circumvent the problem by applying the evaluation methods to a virtual version of the object of interest, the model. Although such models can have any arbitrary complexity to reflect every detail of the real thing, their practical usefulness is limited by the available methods and tools to construct and evaluate them. *Simulation* approaches accept a wide range of model classes but often result in long computation time if information related to rare events is needed. Appropriate actions have to be taken in order to deal with such situations [7, 112]. For the *model-based numerical analysis*, especially methods based on Markovian processes have been successfully adopted. Analysis methods like discrete-time, resp. continuous-time Markov chains (DTMCs, resp. CTMCs) extend the classical deterministic state machines by the concept of state probabilities. This state-oriented approach allows the intuitive modeling of many processes of the real world, especially if the design behind those processes also is state-based (which is, for example, the case for network protocols [89]).

However, these methods are sensitive to the well-known phenomenon called *state space explosion*. For any realistic system, a direct naive representation of the system behavior by a model leads to a state space that is far too large for any analytical treatment. One way to handle this problem is to avoid the state-oriented approach, for example by flow-oriented analysis [3] or simulation [53]. This will, however, not be the topic of this thesis. Instead, we will follow a *decomposition approach* which aims to reduce the number of states in a model. The decomposition approach achieves this by dividing a large model into smaller submodels which can then be independently analyzed. A fixed-point iteration scheme is used to combine the solutions of the submodels in order to compute an approximate solution for the original, overall model. The decomposition and the iteration-based solution approach were successfully applied to various model classes in the past. However, most attempts bore monolithic tool implementations that could not be easily extended or combined with other formalisms or solution methods. As a reaction, tools like SMART [22] have been developed that offer multiple modeling formalisms and permit the easy integration of new solution methods. The Möbius framework and tool [23, 26] can be considered as the next step of this development; it provides an infrastructure for formalisms and solvers with minimal assumptions about their functionality, the supported model classes, and how submodels are combined. But the reduced computational complexity is not the only motivation for the decomposition approach. It also promotes a *component-oriented* way of modeling where the submodels are the components. Similar to other engineering disciplines, one generally prefers tools that allow to build the model of smaller, well-understood components instead of "inventing" a completely new model for each application case. In addition to the better analytical tractability, this simplifies the design of the model and the interpretation of the evaluation results.

It has turned out that many of the arguments stated above are especially applicable to queueing networks. Their building blocks, the queueing stations, are able to directly model many processes that we encounter in the real world. A decomposition at the level of queueing stations is possible for many interesting applications. In addition, the queueing process of a single station has been intensively studied by means of CTMCs and Quasi-Birth-and-Death processes (QBDs) [1, 9, 13, 14, 49, 68]. Hence, the decomposition of queueing networks naturally follows [43, 44, 45, 47, 103, 108].

In this thesis, we introduce a decomposition framework for the analysis of general queueing networks. In the context of this framework, two critical points in the analysis of queueing networks can be identified: (i) the representation of the traffic streams between the queueing stations, and (ii) the employed algorithms for the analysis of the single queueing stations. We will present and discuss different choices for these issues as well as the resulting analysis methods.

Organization of the thesis

In Chapter 2, we first give an intuitive introduction to the decomposition approach by means of general models (which are not necessarily queueing networks). In the same chapter, we show that the restriction to queueing networks allows for an elegant description of the operations that have to be performed in order to analyze a model. This description is then further developed to a decomposition framework for a quite general class of queueing networks. A fundamental question is how the various arrival and service processes and the queueing processes inside a queueing network are mathematically represented. We rely in the following on Markovian Arrival Processes (MAPs), phase-type renewal processes and distributions (PHs) and Quasi-Birth-and-Death processes (QBDs). Chapter 3 introduces the involved mathematical structures and gives an overview of the available solution methods for QBDs.

Over the last decade, traffic measurements have shown the presence of properties such as self-similarity and long-range dependency in network traffic. It has been shown that this can be explained by the heavy-tailedness of many of the involved distributions. In Chapter 4, we describe how hyper-exponential distributions, which are a special case of phase-type distributions, can be fitted to heavy-tail distributed measurement data.

The first applications of our decomposition framework are given in Chapter 5. We illustrate how existing analysis methods like QNA [115, 116] can be elegantly described in the context of the framework. Then we present *FiFiQueues*, our decomposition-based analysis method for queueing networks with queueing stations of finite and infinite capacity. In this chapter, we also discuss the fixed-point behavior of FiFiQueues and various extensions to the original algorithm. The performance of FiFiQueues is evaluated in Chapter 6. The evaluation includes various tests with representative queueing networks and a case study of a web server.

Although most of this thesis concerns open queueing networks, some systems can be more elegantly modeled by *closed* queueing networks. In Chapter 7, we present an approach to analyze a certain class of closed queueing networks by approximations through open queueing networks.

Combining all the previously described results, we have developed the *FiFiQueues network designer*, a tool that comprises the implementation of the FiFiQueues algorithms, a queueing network simulator and a graphical user interface. It is described in Chapter 8.

The algorithms employed by FiFiQueues and related methods are based on PH|PH|1(|K) queues. Naturally, PH renewal processes are not able to represent correlations in the arrival and service processes. The usage of MAP|MAP|1(|K) queues should overcome this problem. However, reduction methods are required to avoid the potential state space explosion. They are presented in Chapter 9 and evaluated in Chapter 10.

We finally conclude this thesis with a summary of the results and with an outlook in Chapter 11.

Chapter 2

Analysis of General Queueing Networks

Queueing networks (QNs) have been used widely for the analysis of performance problems in computer and communication systems. For many classes of queueing networks elegant and efficient solution methods exist. In case the QNs under study are open and contain queueing stations with infinite capacity, i.e., when the number of customers is not *a priori* restricted, product-form results, such as those for Jackson networks [54], can be used. A disadvantage of these results is that they are only valid under a number of restrictions: the service times need to be exponentially distributed when combined with FCFS scheduling, the stations have unbounded buffer capacity, and all arrival processes are Poissonian. These restrictions might not always apply in practice.

The above restrictions have led researchers to search for extensions and approximations. Queueing network models with either finite customer number or with finite buffers (and hence with customer losses) can be analyzed via the numerical solution of the underlying CTMC, although it might lead to CTMCs with very large state spaces [46].

The approach described in this chapter has been motivated by the approximate solution method of large open queueing networks with infinite-buffer stations and FCFS scheduling, as proposed by Kühn [64] and later extended by Whitt [115, 116]. A key issue in their approach is the *approximate* decomposition of the overall model in separate models for each queue.

This chapter is structured as follows: first, we will present the decomposition approach in general in Section 2.1, and specialized to the context of queueing networks in Section 2.2. Then, we describe in Section 2.3 how such networks can be analyzed. Finally, in Section 2.4, we will give a very general class of queueing networks that will form the base of all following chapters in this work. A summary is given in Section 2.5.

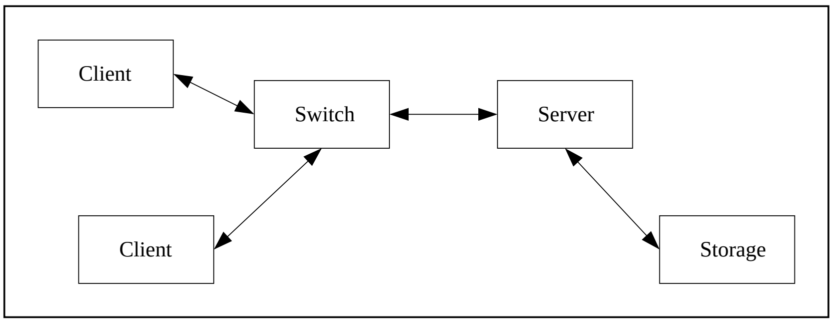


Figure 2.1: Abstract system representation

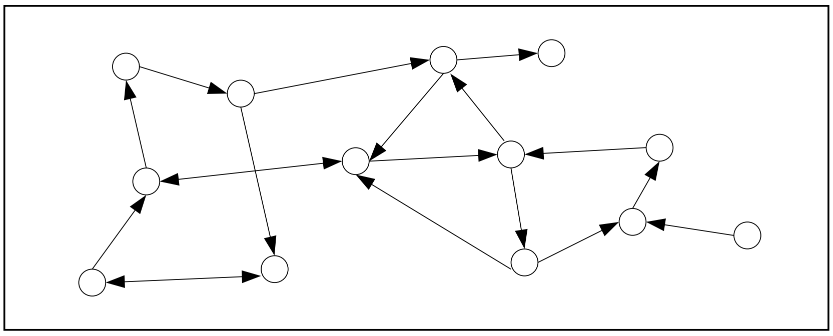


Figure 2.2: Example CTMC

2.1 The decomposition approach

We explain the decomposition approach by an example in Section 2.1.1. A more abstract description of the approach is given in Section 2.1.2. The decomposition is possible for open and for closed models, as explained in Section 2.1.3.

2.1.1 An example system

Figure 2.1 shows a small communication network. Although this picture is clearly not the real system, it is neither a model in the sense as used in the following chapters. The boxes stand for parts of the real system that were too complex to show them in full detail and the lines between them simply describe “logical” connections between those parts. To derive a model from this network the modeler has to choose the — for her purpose — most interesting aspects of the system. The model may be a stochastic Petri net that is evaluated by constructing and analyzing the underlying CTMC. This CTMC may look as the one shown in Figure 2.2. In this figure each circle stands for a possible state of the system and the arrows are transitions from one state to another.

Some analysis methods rely on an analysis of the full state space and hence they suffer from the state space explosion phenomenon: if two (sub-)models A and B with a resp. b possible states are composed to a new model $A \times B$, this new model has $a \cdot b$ possible states (this assumes that there are no mutually exclusive states). Let us assume that we want to model the communication system on the TCP/IP-packet level. Even a very rough estimation shows that each host in the network may take at least some 100 states representing the number of packets in the receiving buffers, the window size of the sender process, etc. With 4 hosts (2 clients, one switch, and one server) our model would already have $100^4 = 10^8$ states. This number of states quickly grows beyond what can be practically handled.

2.1.2 The decomposition approach

The decomposition approach is based on the following idea: If we have two submodels A and B with a resp. b possible states, we avoid to construct and analyze the "product"-model $A \times B$. Instead we do the following:

1. We assume that the system has the structure $B(A)$ instead of $A \times B$, i.e., that in the resulting composition the submodel B depends on A but not vice versa.
2. We analyze model A independently of B and summarize its behavior in some so-called *descriptor* d_A.
3. The descriptor d_A is used to parameterize model B and we analyze the new model $B(d_A)$ with $b(d_A)$ states instead of $B(A)$ with $a \cdot b$ states.
4. Now, we know the behavior of A combined with B. The global behavior of the system can then be derived.

We do the following observations:

- The decomposition approach reduces the number of states to analyze from $a \cdot b$ to a, and, after that, $b(d_A)$.
- The assumption in step 1 is certainly not always true. However, we will see in Section 2.3 that it can be weakened for some networks with cyclic structure.
- The analysis of A and $B(d_A)$ instead of $B(A)$ only makes sense if $B(d_A)$ has fewer states than $B(A)$, or, more colloquial, "d_A really is a reduction of A's behavior".

In most cases we have to simplify the original model to fulfill the assumption made in step 1 before we can start to decompose it. However, the main problem is to find a "good" descriptor d_A of A's behavior: it has to be less complex than A but it should carry enough information about A in order to obtain correct results from $B(d_A)$. Clearly, the decomposition approach only provides an approximation to the original

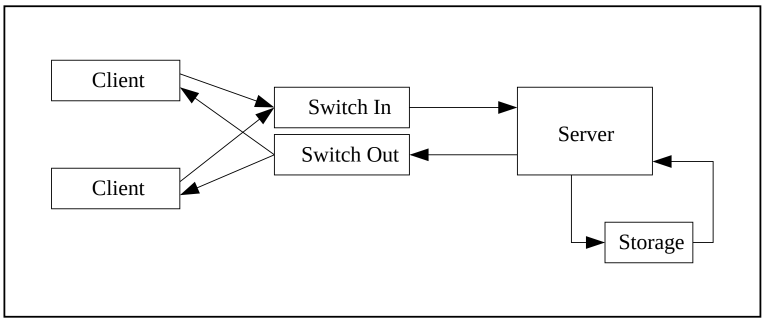


Figure 2.3: Decomposition of the original system

model if d_A is not equivalent to A. Since, in general, the original model already is a simplification of the real system, the decomposition approach adds another source of error to the results.

In more complex systems with more than two interacting submodels, a submodel both receives one (or more) descriptors (called input or arrival descriptors) and creates one (or more) descriptors (called output or departure descriptors). Hence, we have to perform the three steps to analyze an arbitrary submodel S with input descriptors $in_1, in_2, \ldots$ and output descriptors $out_1, out_2, \ldots$, as follows:

1. construct the submodel $S(in_1, in_2, \ldots)$,
2. analyze $S(in_1, in_2, \ldots)$, and
3. compute $out_1, out_2, \ldots$.

Note that we still assume that the system has the structure $B(A)$ (see above). Hence, in order to analyze the whole system, we start with the analysis of the submodels that do not depend on any input descriptors (or where all input descriptors are known because, e.g., they are part of the model specification). Then, we continue with the submodels that only depend on the previously analyzed submodels, and so forth until all submodels have been analyzed.

Eventually, let us assume that we have applied the decomposition approach to the original model of our example communication system. A possible resulting model is shown in Figure 2.3. The boxes and arrows have the following semantics: boxes stand for the submodels and the arrows express dependencies between submodels, i.e., each arrow implies that a descriptor is transfered from one submodel to another in order to analyze the complete model. Note the following two facts:

1. It is not needed that all submodels are of the same type. For example the server submodel may be a stochastic Petri net whereas the storage submodel is described by a queueing station. The same is true for the type of the descriptors: it may vary from arrow to arrow. However, combining different

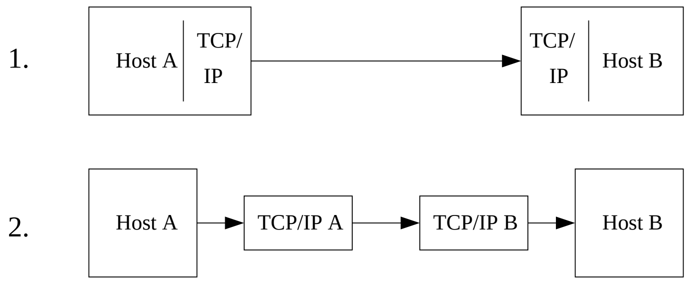


Figure 2.4: Decomposition alternatives

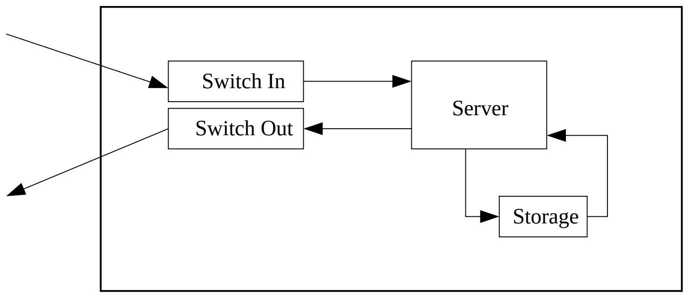


Figure 2.5: Open network

classes of submodels often requires conversion operations on the descriptors, or sometimes is not possible at all, since each submodel only accepts specific types of input descriptors.

2. There is no unique decomposition of a model. In Figure 2.4 we have shown two alternative models for the TCP/IP connection between two hosts. In alternative 1 the submodels of both hosts also contain the TCP/IP stacks whereas in alternative 2 a dedicated submodel is used for this.

2.1.3 Open and closed network models

So far, our examples have shown *closed* network models where the submodels only are mutually dependent. In fact, this thesis mainly concerns *open* networks as shown in Figure 2.5. In such models, the behavior of the submodels also depends on an "outside" world. Of course, the decomposition method can also be applied to open models: the influence on the model from the outside world has to be represented by descriptors, too. Note that, unlike the other descriptors, these descriptors are part of the model specification and do not need to be computed by the analysis method.

2.2 Application to queueing networks

In the following sections, we specialize the decomposition approach to queueing networks. In Section 2.2.1, we explain the decomposition of such networks and the analysis of the resulting submodels. We will see that the decomposition approach imposes important restrictions to queues with finite capacity. They are discussed in Section 2.2.2.

2.2.1 Decomposition of queueing networks

In the context of queueing networks the decomposition will be done at the level of individual queueing stations, i.e., each submodel describes a single queueing station. A station specification consists of two components:

1. a queue with finite or infinite capacity,
2. one or more service entities that serve the jobs (served jobs leave the queue),

and two policies:

1. a policy that handles incoming jobs if the queue is full (only for finite queues),
2. a scheduling policy that describes how the service stations fetch new jobs from the queue.

The descriptor $desc_{i,j}$ describes the traffic stream from queueing station i to station j. Descriptors associated with traffic streams will be called *traffic descriptors* in the following. In case of an open network, we also have traffic descriptors from the outside world to the queueing stations, and vice versa. We will represent the outside world by a "virtual" station *ext* and denote the associated descriptors $desc_{ext,i}$ resp. $desc_{i,ext}$ for station i. Since most analysis methods require that a queueing station has exactly one arrival descriptor and one departure descriptor, a *traffic merging* (or *traffic superpositioning*) and a *traffic splitting* step are required. The traffic merging step merges for a station its arrival descriptors into a single overall arrival descriptor whereas the traffic splitting step splits the overall departure descriptor into the required number of departure descriptors.

In this thesis we mainly use stochastic analysis methods that analyze the queueing stations in terms of arrival and departure point-processes where the inter-event time of the processes represent the inter-arrival time, respectively, inter-departure time of jobs. Thus, to analyze a queue we have to perform the following steps (see Figure 2.6):

1. traffic merging;
2. construct arrival process from arrival traffic descriptor;
3. construct queueing process;

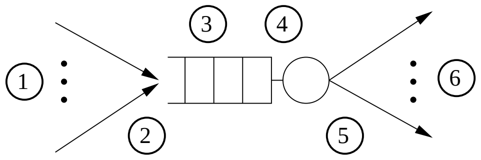


Figure 2.6: Steps in the analysis of a queueing station

4. analyze queueing process and compute departure process;

5. convert the departure process into the departure traffic descriptor;

6. traffic splitting.

In practice, the whole procedure is very flexible. In the following we give some examples how the different steps can be varied:

- **direct merging and splitting:** the merging step directly constructs the arrival process from the arrival descriptors without the intermediate superpositioned descriptor. For example, methods like the stationary-interval method (see Section 5.5.2) first construct intermediate arrival processes from the arrival descriptors and then perform the traffic merging directly on the intermediate arrival processes in order to obtain the overall arrival process representation.

  Similarly, the splitting step may directly operate on the departure process. Thus, step 1 and 2, resp. 5 and 6 are merged.

- **variable departure descriptors:** the conversion procedure that converts the departure process into the departure descriptor may choose the form of the descriptor depending on the structure of the point process. For example, the complexity of the traffic descriptor can be adapted to the complexity of the point process. This approach is followed by the methods presented in Chapter 9.

- **variable analysis method:** depending on the properties of the arrival descriptors a specific method is selected to analyze the station. For example, in [47], traffic descriptors with squared coefficient of variation (one of the parameters of the employed descriptors) smaller than 1.0 lead to an analysis method different from the one used in the case where that coefficient is larger than 1.0.

Even the decomposition scheme can be varied: for example, one may decide to decompose only parts of the network down to individual queueing stations whereas other parts are analyzed as a whole without decomposition. We will not pursue this in this thesis.

2.2.2 Finite queues and blocking

Although the decomposition approach is very flexible it should be noted that it imposes an important restriction to queues with finite capacity. Let us take a look at the following situation: a queue A has successfully served a job and now wants to send the job to the next queue B. If the next queue is finite it may happen that it has reached its full capacity and thus cannot accept the job. Different mechanisms have been developed in the past to handle such a situation [86]:

1. the job is simply discarded (this is called *communication blocking*),

or one of the following blocking mechanisms is used:

2. the job is sent back to queue A for repetitive service (RS). After that, the job is either resent to the same queue B (*repetitive service with fixed destination, RS-FD*) or a new destination is determined (*repetitive service with random destination, RS-RD*).
3. queue A stops operating until queue B has free capacity. This behavior may lead to a deadlock in cyclic networks.

Alternatively, there are mechanisms where queue A checks the capacity of B already while it is serving the job (so-called *blocked-before-service* mechanisms).

Since the decomposition approach analyzes all queueing stations separately, the only possible strategy is to discard the job (strategy 1) because no information about free queueing capacities can be exchanged between the queues. However, the restriction to the first strategy can be circumvented in some cases by means of more complex decomposition algorithms. For example, it is sometimes possible to transform the network with blocking into an equivalent non-blocking network. Another approach is to keep queues that exchange information about free queueing capacities in one submodel.

2.3 Analysis of decomposed networks

In the previous section we have shown how a single queueing station that is part of a decomposed queueing network can be analyzed. It is quite obvious how the procedure can be iteratively applied to all stations of a simple tandem queueing network. But in Section 2.1 we have announced that even networks with cyclic structure can be analyzed by the decomposition approach. In the following sections we will explain how complex networks can be analyzed. We describe the analysis of open networks without feedback in Section 2.3.1. Possible parallelizations of the involved computations are discussed in Section 2.3.2. An algorithm for the analysis of open networks with feedback is presented in Section 2.3.3. Finally, closed networks are briefly discussed in Section 2.3.4.

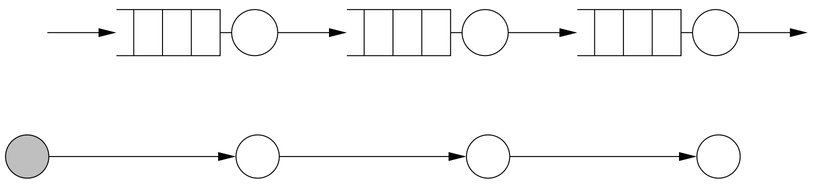


Figure 2.7: Simple queueing network and its graph

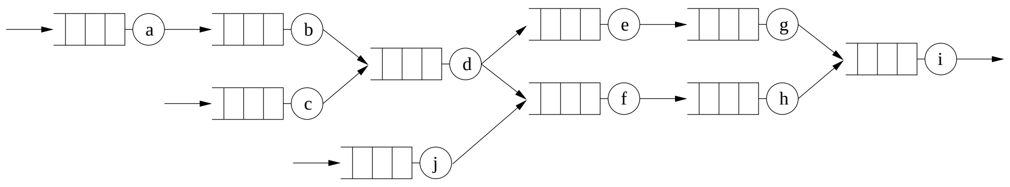


Figure 2.8: Queueing network without feedback

2.3.1 Open networks without feedback

As first step, we move to a more abstract view of queueing networks. We represent them by graphs $G(V, E)$ where the set of vertices V contains the queueing stations and the relation $E(n, m)$ holds if there is a direct connection from station n to station m. If the network is open, we introduce an *external node* that represents the source of the external traffic. Figure 2.7 shows a simple open tandem network and its corresponding graph (the external node is shaded). Note that the connections *from* the nodes *to* the external world are not shown in the graph because they will not influence the analysis.

Queueing networks without feedback have graphs without cycles. Such networks can be analyzed in a single pass through all queueing stations. Figure 2.8 shows a quite complex open network without feedback. The corresponding graph can be seen in Figure 2.9.

Obviously, in our example queue d depends on b and c, so we first have to process them so that all arrival descriptors of d are known. On the other hand, it is not

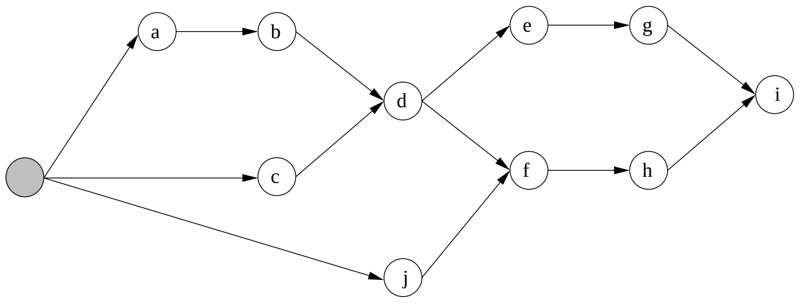


Figure 2.9: Graph of queueing network without feedback

precedence	nodes
0	external node
1	a, c, j
2	b
3	d
4	e, f
5	g, h
6	i

Table 2.1: Order of analysis for the network without feedback

so important whether we first process a, c or j — all three queues only depend on traffic descriptors that they receive from external sources.

Order of analysis

It is quite easy to compute the order of processing for the queueing stations. For this purpose we define the function order : $V \to \mathbb{N}$ that assigns to each node its processing precedence. The external node does not depend on other nodes, hence it is processed first:

$$\text{order(external node)} = 0$$

A queueing station $n \in V$ is assigned a higher order number than its predecessors in the graph since it can be only analyzed if its arrival descriptors are known:

$$\text{order}(n) = 1 + \max_{m} \{\text{order}(m) \mid E(m,n) \text{ and } m \in V\}\,.$$

Table 2.1 shows the results for the network of Figure 2.8, sorted by precedence. To analyze the whole network we first analyze the nodes with precedence 1, then the nodes with precedence 2, etc., until all nodes have been processed.

2.3.2 Parallelization

In a multi-processor environment a speed-up can be achieved in two different ways:

1. The analysis of the single node is parallelized. For example, if the node is evaluated by a replication-based simulation [92], the different replications can be performed in parallel.
2. The analysis of the network is parallelized, i.e., one tries to analyze two or more nodes at the same time. A simple parallelization approach can be discovered if we look at the precedences of the nodes. Obviously, nodes with the same precedence are independent and hence can be analyzed in parallel. However, in

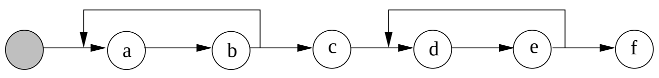


Figure 2.10: Graph of queueing network with feedback

general, this approach does not yield good speed-ups because the complexity of the analysis may considerably vary in time for each node. Better speed-ups are obtained by a dynamic scheduling: a node n that is ready for processing (i.e., all arrival descriptors of n are known) is placed into a list L where it can be picked up by an idle CPU.

Both approaches can be mixed. As an example, consider a network with 2 nodes x and y waiting for processing where node x should be evaluated by a simulation with 5 replications and node y should be analyzed, e.g., by a matrix-geometric method. For optimal speed-up we can place 6 elements into L:

$$L := \{\text{repl. 1 of } x, \text{repl. 2 of } x, \text{repl. 3 of } x, \text{repl. 4 of } x, \text{repl. 5 of } x, \text{analysis of } y\}\,.$$

2.3.3 Open networks with feedback

The graph of an open network with feedback contains cycles. Figure 2.10 shows such a graph.

Analysis of the network

Our analysis of open queueing networks with feedback relies on the fixed-point iteration algorithm presented in [44, 113]:

```
1 initialize all traffic descriptors $desc^{(0)}_{i,j}$:
2         set $desc^{(0)}_{ij}$ to the null value if $i \neq ext$
3         set $desc^{(0)}_{ij}$ to the specified value if $i = ext$
4 $n := 0$
5 do
6         $n := n + 1$
7         analyze each queueing station $i$ and compute $desc^{(n)}_{i,j}$ for all nodes $j$.
8 while $dist(desc^{(n)}, desc^{(n-1)}) > \varepsilon$
```

In each iteration a new set of traffic descriptors $desc^{(n)} = \{desc^{(n)}_{i,j} | i, j\}$ is computed. The algorithm stops when the distance $dist(desc^{(n-1)}, desc^{(n)})$ between two successive sets of descriptors is smaller or equal than a given threshold ε. We have chosen the Euclidean distance as distance function in the following chapters. Descriptors set to the *null* value in line 2 are ignored in line 7. This value indicates that no information about the descriptors $desc_{ij}$ with $i \neq ext$ is available when the algorithm starts.

Order of analysis

From the example in Figure 2.10 we can see that it is not necessary to analyze all queues in each round of the iteration. Obviously, the optimal way is to analyze node a and b in a loop until stable results are obtained, then analyze node c, analyze d and e again in a seperate loop and finally analyze node f. An implementation of this intuitive approach is given in [47]: the algorithm presented there identifies the loop-structures (overlapping loops are considered as one large loop) in the graph and calls the iteration scheme for each of them. Nodes that do not belong to a loop are only visited once.

It should be noted that it is not possible to provide the optimal order[1] for arbitrary node analysis algorithms and arbitrary networks (especially if they contain overlapping loops) since the analysis of a queueing station requires a non-linear, sometimes non-continuous, transformation of the arrival descriptor into the departure descriptor. For example, even a small change to an arrival descriptor may cause the analysis algorithm to switch to another type of queueing station model. In fact, in general it is not known whether the searched fixed point always exists, is unique or can be found. However, in our experiments with the FiFiQueues network analyzer (see Chapter 5) the computation always terminated. This indicates that the FiFiQueues algorithm is, as far as we have experienced, not sensitive to the problem. Some preliminary results concerning its iteration behavior will be given in Section 5.4.

2.3.4 Closed networks

In closed queueing networks a global dependency between all nodes is added: the (average) number of jobs in the system has to be equal to some finite number K. Obviously, the methods for open networks cannot be directly applied here. We will discuss in detail in Chapter 7 how the decomposition approach can be extended to closed networks.

[1]the order that leads to the smallest computation time or memory usage.

2.4 A decomposition framework for queueing networks

From the previous sections we can derive a decomposition framework for queueing networks that is able to cover different types of traffic descriptors and analysis methods. Its base characteristics are:

1. The stations are supposed to be independent of each other, apart from the interchanged traffic streams and the job number restriction for closed networks.
2. A fixed-point iteration is employed to solve the network model.

In addition to these characteristics, we limit the supported model class to networks with constant Markovian routing, i.e., the stream of jobs leaving a station is split according to constant probabilities which only depend on the station. Everything else, especially the type of the employed traffic descriptors, is up to the individual analysis methods. Hence, the specification of a queueing network model consists of the following information:

- a routing matrix $\mathbf{\Gamma} = (r_{ij})$ of size $n \times n$ where n is the number of queueing stations and r_{ij} specifies the routing probability from station i to station j,
- the description for each station as required by the analysis method, e.g., the service process, the queueing capacity (if finite), etc.,
- the descriptors of the external arrival traffic for each station (only for open networks).

We also support two post-processing steps that are performed after the fixed-point iteration. They allow to compute additional performance measures:

1. computation of node specific results (e.g., mean queue length),
2. computation of network wide results (e.g., total network throughput).

Hence, when we discuss a particular analysis method in the following chapters, we usually address the following topics in order to completely describe the method:

1. the type of the employed traffic descriptors,
2. the traffic merging (superpositioning) operation (step 1 in Figure 2.6),
3. the traffic splitting operation (step 6 in Figure 2.6),
4. the service operation (steps 2–5 in Figure 2.6),

5. the computation of node specific results,
6. the computation of network wide results.

The complexity of the involved operations is addressed, too.

2.5 Summary and conclusions

In this chapter we have presented the decomposition approach and specialized it to queueing networks. The decomposition is done at queueing station level, i.e., each submodel describes a single queueing station. During the analysis, traffic descriptors are exchanged between the submodels that represent the streams of jobs flowing between the queueing stations. Networks without feedback can be analyzed in a single pass whereas networks with feedback require the usage of a fixed-point iteration. We have shown that the analysis of a queueing station can be divided into six fundamental steps that describe how the incoming traffic streams are merged, processed by the queueing station, and then split into the outgoing traffic streams. This abstract view has allowed us to create a general decomposition framework for queueing networks which we will use in the following chapters.

Chapter 3

Markovian Arrival Processes, Phase-Type Renewal Processes, and Quasi-Birth-and-Death Processes

In this chapter we introduce the fundamental mathematical structures and notations that this thesis is based on. We begin with an important class of stochastic processes, the Markovian Arrival Processes (MAP) in Section 3.1. Phase-type (PH) renewal processes, which can be seen as special cases of MAPs, are introduced in Section 3.2. The queueing processes that we will discuss in the next chapters have underlying Markov chains that belong to the well-known class of continuous-time Quasi-Birth-and-Death (QBD) processes. We give the formal definition of QBD processes as well as methods to compute their steady-state solution. We first discuss infinite QBDs in Section 3.3 and continue with finite QBDs in Section 3.4. In Section 3.5 we comment on the performance of the solution methods.

3.1 Markovian Arrival Processes (MAPs)

3.1.1 Definition and notation

Markovian Arrival Processes (MAPs) [74, 75, 84] belong to the general class of point processes and can be seen as special cases of Matrix Exponential Point Processes (which, in turn, form a subset of the class of Semi-Markov Processes [46]). MAPs cover many interesting processes including the Markov-Modulated Poisson Processes (MMPPs) [35] and the phase-type (PH) renewal processes (see below).

A MAP can be described by a finite irreducible continuous-time Markov chain (CTMC) with generator matrix $\mathbf{Q}$ where some transitions are "marked". Every time when the process passes through such a marked transition an event is triggered. The time instants of these events form the point process. We follow the notation of [76] in the following and split the generator matrix into two matrices $\mathbf{Q_0}$ and $\mathbf{Q_1}$ as follows:

$$\mathbf{Q_0} = \begin{pmatrix} -q_1 & q_{12} & \cdots & q_{1m} \\ q_{21} & -q_2 & \cdots & q_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ q_{m1} & q_{m2} & \cdots & -q_m \end{pmatrix}, \quad \mathbf{Q_1} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mm} \end{pmatrix},$$

with $\mathbf{Q_0} + \mathbf{Q_1} = \mathbf{Q}$ where $q_i = a_{ii} + \sum_{j=1, j\neq i}^{m}(q_{ij} + a_{ij})$. The elements of the matrix $\mathbf{Q_1}$ give the transition rates of the marked transitions.[1] In the following, we denote a MAP by the pair $(\mathbf{Q_0}, \mathbf{Q_1})$ and call m the *size* of the MAP.

3.1.2 Characteristics

Some general results of the Markov-modulated Poisson process [35] can be easily adapted to the MAP. In order to compute the behavior of a MAP $(\mathbf{Q_0}, \mathbf{Q_1})$ we first need to choose the initial probability vector $\mathbf{p}$ of the MAP. In analogy to phase-type renewal processes we start the MAP at an "arbitrary" arrival epoch by choosing

$$\mathbf{p} = \frac{1}{\boldsymbol{\pi}\mathbf{Q_1}\mathbf{1}}\boldsymbol{\pi}\mathbf{Q_1},$$

where $\boldsymbol{\pi}$ is the steady-state probability vector of the MAP, i.e., $\boldsymbol{\pi}(\mathbf{Q_0} + \mathbf{Q_1}) = \mathbf{0}$. The thus-obtained process is said to be *interval-stationary*. The inter-arrival time distribution function of the interval-stationary process is given by

$$F(t) = 1 - \mathbf{p}\exp(\mathbf{Q_0}t)\mathbf{1}, \tag{3.1}$$

which leads to the following expression for the kth moment of the inter-arrival time:

$$\mathrm{E}[T^k] = k!\mathbf{p}(-\mathbf{Q_0})^{-(k+1)}\mathbf{Q_1}\mathbf{1}. \tag{3.2}$$

Hence, the first moment of the inter-arrival time is given by

$$\mathrm{E}[T] = \frac{1}{\boldsymbol{\pi}\mathbf{Q_1}\mathbf{1}}\boldsymbol{\pi}\mathbf{Q_1}(-\mathbf{Q_0})^{-2}\mathbf{Q_1}\mathbf{1}.$$

This equation can be further simplified by using the equations $\boldsymbol{\pi}\mathbf{Q_1} = -\boldsymbol{\pi}\mathbf{Q_0}$ and $\mathbf{Q_1}\mathbf{1} = -\mathbf{Q_0}\mathbf{1}$ which follow from the definition of π, respectively, from the fact that

[1]This definition allows the following interpretation of the matrices $\mathbf{Q_0}$ and $\mathbf{Q_1}$: passing through a transition given as entry of $\mathbf{Q_1}$ triggers the generation of *one* event. Batch Markovian Arrival Processes (BMAPs) generalize this viewpoint by introducing matrices $\mathbf{Q_i}$ with $i > 1$ whose entries describe transitions with batch arrivals of size i.

$\mathbf{Q_0} + \mathbf{Q_1}$ is a stochastic matrix. We find that the arrival rate λ of a MAP (the inverse of the first moment) is

$$\lambda = \boldsymbol{\pi}\mathbf{Q_1}\mathbf{1}$$

which yields

$$\mathrm{E}[T^k] = \frac{k!}{\lambda}\boldsymbol{\pi}(-\mathbf{Q_0})^{-(k-1)}\mathbf{1}.$$

Let T_i be the time between the ith and the $(i+1)$st arrival in a MAP. Then, the autocovariance function $R(k)$ for T_1 and T_{k+1} with $k \geq 1$ is given by

$$\begin{aligned} R(k) &= \mathrm{E}\left[(T_1 - \mathrm{E}[T_1])(T_{k+1} - \mathrm{E}[T_{k+1}])\right] \\ &= \mathbf{p}(-\mathbf{Q_0})^{-2}\mathbf{Q_1}\left\{\left[(-\mathbf{Q_0})^{-1}\mathbf{Q_1}\right]^{k-1} - \mathbf{1}\mathbf{p}\right\}(-\mathbf{Q_0})^{-1}\mathbf{1}. \end{aligned}$$

The limiting index of dispersion I of a MAP is given by [47]

$$I = \lim_{t\to\infty}\frac{\mathrm{Var}[N(t)]}{E[N(t)]} = 1 + 2\left(\lambda - \frac{1}{\lambda}\boldsymbol{\pi}\mathbf{Q_1}(\mathbf{Q_0} + \mathbf{Q_1} + \mathbf{1}\boldsymbol{\pi})^{-1}\mathbf{Q_1}\mathbf{1}\right),$$

where $N(t)$ is the counting process of the MAP.

3.1.3 Superposition and Markovian splitting

The class of MAPs is closed under superposition and Markovian splitting. The superposition of two MAPs $(\mathbf{A_0}, \mathbf{A_1})$ and $(\mathbf{B_0}, \mathbf{B_1})$ is a new MAP $(\mathbf{C_0}, \mathbf{C_1})$ with

$$\mathbf{C_0} = \mathbf{A_0} \oplus \mathbf{B_0}, \quad \mathbf{C_1} = \mathbf{A_1} \oplus \mathbf{B_1},$$

where $\mathbf{L}\oplus\mathbf{M} = \mathbf{L}\otimes\mathbf{I} + \mathbf{I}\otimes\mathbf{M}$, and $\otimes$ is the Kronecker product operator (also known as tensor or matrix direct product operator).

The Markovian splitting of a MAP $(\mathbf{A_0}, \mathbf{A_1})$ with probability r gives two MAPs $(\mathbf{B_0}, \mathbf{B_1})$ and $(\mathbf{C_0}, \mathbf{C_1})$ with

$$\begin{aligned} (\mathbf{B_0}, \mathbf{B_1}) &= (\mathbf{A_0} + (1-r)\mathbf{A_1}, r\mathbf{A_1}), \\ (\mathbf{C_0}, \mathbf{C_1}) &= (\mathbf{A_0} + r\mathbf{A_1}, (1-r)\mathbf{A_1}). \end{aligned}$$

3.1.4 Markov-Modulated Poisson Processes (MMPPs)

The MMPP is the doubly stochastic Poisson process whose arrival rate depends on the state of an irreducible Markov process. Thus, MMPPs can be seen as MAPs where the matrix $\mathbf{Q_1}$ is restricted to the form

$$\begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{mm} \end{pmatrix}.$$

3.2 Phase-type (PH) renewal processes

3.2.1 Definition and notation

A continuous phase-type renewal process can be seen as a special MAP $(\mathbf{A}, \mathbf{A^0}\boldsymbol{\alpha})$ where $\mathbf{A^0}$ is a $n \times 1$ column vector with entries and $\boldsymbol{\alpha}$ is a $1 \times n$ row probability vector. Consequently, it holds $\mathbf{A^0} = -\mathbf{A}\mathbf{1}$.

We adopt the notation of [85] and denote PH renewal processes by the pair $(\boldsymbol{\alpha}, \mathbf{A})$ which can be interpreted as follows: the $n \times n$ matrix $\mathbf{A}$ describes the n transient states of a CTMC with $n+1$ states. The last state $n+1$ is an absorbing state and any transition (given by $\mathbf{A^0}$) from the transient state to the absorbing state will trigger an arrival. After the arrival, the process will restart in the transient state i with probability α_i. Furthermore, PH inter-event time distributions form a dense subset of all distributions with support on $[0; \infty)$, i.e., any distribution can be approximated arbitrarily closely by a PH distribution [58].

3.2.2 Inter-event time characteristics

Obviously, the vector $\boldsymbol{\alpha}$ of the PH renewal process $(\boldsymbol{\alpha}, \mathbf{A})$ is identical to the interval-stationary probability vector $\mathbf{p}$ of the corresponding MAP. Hence, expressions for the distribution function of the inter-event time and the k-th moment directly follow from Equations (3.1) and (3.2) and we have

$$F(t) = 1 - \boldsymbol{\alpha} \exp(\mathbf{A}t)\mathbf{1},$$

respectively

$$\mathrm{E}[T^k] = k!\boldsymbol{\alpha}(-\mathbf{A})^{-k}\mathbf{1}.$$

Note that the matrix $\mathbf{A}$ is nonsingular, so that all moments are finite. From this follows that the MAP $(\mathbf{Q_0}, \mathbf{Q_1})$ and the PH renewal process $(\mathbf{p}, \mathbf{Q_0})$ have the same inter-event time distribution.

3.2.3 Superposition and Markovian splitting

The superposition of two PH renewal processes $(\boldsymbol{\alpha}, \mathbf{A})$ and $(\boldsymbol{\beta}, \mathbf{B})$ is a MAP $(\mathbf{C_0}, \mathbf{C_1})$ with

$$\mathbf{C_0} = \mathbf{A} \oplus \mathbf{B}, \quad \mathbf{C_1} = \mathbf{A^0}\boldsymbol{\alpha} \oplus \mathbf{B^0}\boldsymbol{\beta}.$$

Note that the class of PH renewal processes is not closed under superposition.

The Markovian splitting of a PH renewal processes $(\boldsymbol{\alpha}, \mathbf{A})$ with probability r gives two PH renewal processes $(\boldsymbol{\alpha}, \mathbf{A} + (1-r)\mathbf{A^0}\boldsymbol{\alpha})$ and $(\boldsymbol{\alpha}, \mathbf{A} + r\mathbf{A^0}\boldsymbol{\alpha})$.

3.3 Infinite QBDs

This section is based on Chapter 4 of [88]. Note that we use a simplified notation.

3.3.1 Definition

QBD processes [85] can be described as a generalization of the queueing process of M|M|1 queueing stations. In the underlying Markov chain of such a queue we can identify an infinite number of states where state i describes that i jobs are in the system. The transition from state i to $i+1$ resp. from $i+1$ to i is marked by the arrival rate resp. the service rate of the queueing station.

In QBDs, these states are replaced by so-called *levels*: level i still stands for i jobs in system but in QBDs each level may consist of more than one state. Usually, a two-dimensional addressing scheme is used for the states where (i, j) addresses state j of level i. Note that in the QBD the number of levels is unbounded whereas the number of states per level is required to be finite. Moreover, the levels $1, 2, \ldots$ (the *repeating levels*) have to contain the same number of states N. Level 0 is called *boundary level* and may contain a different number of states N_0.

In QBDs, two adjacent levels i and $i{+}1$ are not connected by one single transition. Instead, arbitrary transitions between the states of two adjacent levels and between states of the same level are allowed. Consequently, the transition rates are specified by matrices:

- the entry (i, j) of the $N_0 \times N$ matrix $\mathbf{B_{0,1}}$ gives the transition rate from state $(0, i)$ to state $(1, j)$. The opposite direction (from level 1 to level 0) is given similarly by the $N \times N_0$ matrix $\mathbf{B_{1,0}}$.

- the entry (i, j) of matrix $\mathbf{A_0}$ gives the transition rate from state (l, i) to state $(l + 1, j)$ where $l = 1, 2, \ldots$. The opposite direction (from level $l + 1$ to l) is given by matrix $\mathbf{A_2}$. Both matrices are of size $N \times N$.

- transitions inside level 0 are specified by the $N_0 \times N_0$ matrix $\mathbf{B_{0,0}}$. Entry (i, j) gives the transition rate from state $(0, i)$ to state $(0, j)$. Correspondingly, transitions inside repeating level l (with $l = 1, 2, \ldots$) are specified by the $N \times N$ matrix $\mathbf{A_1}$.

As can be seen, all repeating levels have a similar transition structure. The above described matrices directly lead to the generator matrix of the QBD Markov chain. If we sort the states lexicographically, i.e., in the sequence

$$(0, 1), \ldots, (0, N_0), (1, 1), \ldots, (1, N), (2, 1), \ldots$$

we obtain the tri-diagonal block generator matrix $\mathbf{Q}$ of infinite size:

$$\mathbf{Q} = \begin{pmatrix} \mathbf{B_{0,0}} & \mathbf{B_{0,1}} & & & & \\ \mathbf{B_{1,0}} & \mathbf{A_1} & \mathbf{A_0} & & & \\ & \mathbf{A_2} & \mathbf{A_1} & \mathbf{A_0} & & \\ & & \mathbf{A_2} & \mathbf{A_1} & \mathbf{A_0} & \\ & & & \ddots & \ddots & \ddots \end{pmatrix}. \tag{3.3}$$

3.3.2 Steady-state solution

The infinite steady-state probability vector $\mathbf{v}$ of the QBD Markov chain with generator matrix $\mathbf{Q}$ fulfills the global balance equation

$$\mathbf{v} \cdot \mathbf{Q} = \mathbf{0}, \tag{3.4}$$

and the normalization condition

$$\mathbf{v} \cdot \mathbf{1} = 1. \tag{3.5}$$

In the following we write $\mathbf{v_0}$ for the vector $(v_1, \ldots, v_{N_0})$ which contains the steady-state probabilities for the states of level 0 and we write $\mathbf{v_i}$ for the vector $(v_{N_0+1+(i-1)\cdot N}, \ldots, v_{N_0+i\cdot N})$ which contains the steady-state probabilities of level $i = 1, 2, \ldots$. With this notation we can rewrite Equations (3.4) and (3.5) as

$$\begin{aligned} \mathbf{v_0 B_{0,0}} + \mathbf{v_1 B_{1,0}} &= \mathbf{0}, & & (3.6)\\ \mathbf{v_0 B_{0,1}} + \mathbf{v_1 B_{1,1}} + \mathbf{v_2 A_2} &= \mathbf{0}, & & (3.7)\\ \mathbf{v_i A_0} + \mathbf{v_{i+1} A_1} + \mathbf{v_{i+2} A_2} &= \mathbf{0}, \qquad \text{for } i = 1, 2, \ldots, & & (3.8)\\ \sum_{i=0}^{\infty} \mathbf{v_i 1} &= 1. & & (3.9) \end{aligned}$$

The regular structure of Equation (3.8) is the key to the efficient solution of the QBD. Two different classes of solution techniques can be distinguished: matrix-geometric solution methods and transform methods. We will describe them briefly in the following.

3.3.3 Matrix-geometric solution methods

The main idea in this class of solution methods is that the solution vector $\mathbf{v}$ has a matrix-geometric form, i.e., it exists a matrix $\mathbf{R}$ of size $N \times N$ with

$$\mathbf{v_i} = \mathbf{v_1 R}^{i-1}, \qquad i = 1, 2, \ldots \tag{3.10}$$

In [85], it is shown that $\mathbf{R}$ is the entry-wise smallest non-negative solution of the quadratic matrix equation

$$\mathbf{A_0} + \mathbf{R A_1} + \mathbf{R}^2 \mathbf{A_2} = \mathbf{0}. \tag{3.11}$$

The methods described in the following try to solve this equation as efficient as possible. Once $\mathbf{R}$ has been determined, the complete stationary vector $\mathbf{v}$ can be computed using Equations (3.6)–(3.10).

The Successive Substitution method

This method, originally suggested in [85], transforms Equation (3.11) into the assignment:

$$\tilde{\mathbf{R}} := -(\mathbf{A_0} + \tilde{\mathbf{R}}^2 \mathbf{A_2}) \mathbf{A_1}^{-1}.$$

It is shown in [85] that, starting with $\tilde{\mathbf{R}} := \mathbf{0}$, $\tilde{\mathbf{R}}$ converges monotonically to the desired matrix $\mathbf{R}$ when the assignment is successively repeated. This approach is very easy to implement but it converges very slowly if a reasonable accuracy is desired. An improvement of this algorithm has been suggested in [65] which slightly reduces the required number of floating point operations per iteration.

The Logarithmic Reduction (LR) method

The LR method is the most popular method for the solution of QBDs. It has been proposed in [66] and focuses on the solution of the equation

$$\mathbf{A_0} + \mathbf{A_1}\mathbf{G} + \mathbf{A_2}\mathbf{G}^2 = \mathbf{0}, \tag{3.12}$$

which is "symmetric" to Equation (3.11). After $\mathbf{G}$ has been computed the matrix $\mathbf{R}$ is given by

$$\mathbf{R} = -\mathbf{A_0}(\mathbf{A_1} + \mathbf{A_0}\mathbf{G})^{-1}. \tag{3.13}$$

The matrix $\mathbf{G}$ can be interpreted as follows: entry (i, j) gives the probability that starting from state $(2, i)$ the QBD will finally enter level 1 in state $(1, j)$. The LR method approximates $\mathbf{G}$ by first assuming that no level above level $k = 2$ is visited by the QBD. The key point is that the algorithm is able to double this limit k in each iteration which results in a quadratic convergence speed. The complete algorithm in pseudo code looks like:

```
$\mathbf{B_0} := -\mathbf{A_1}^{-1}\mathbf{A_0}, \mathbf{B_2} := -\mathbf{A_1}^{-1}\mathbf{A_2}$
$\mathbf{G} := \mathbf{B_2}, \mathbf{T} := \mathbf{B_0}$
while $||\mathbf{1} - \mathbf{G}\mathbf{1}||_\infty \geq \varepsilon$
    $\mathbf{D} := \mathbf{I} - \mathbf{B_0}\mathbf{B_2} - \mathbf{B_2}\mathbf{B_0}$
    $\mathbf{B_0} := \mathbf{D}^{-1}\mathbf{B_0}^2, \mathbf{B_2} := \mathbf{D}^{-1}\mathbf{B_2}^2$
    $\mathbf{G} := \mathbf{G} + \mathbf{T}\mathbf{B_2}, \mathbf{T} := \mathbf{T}\mathbf{B_0}$
end
$\mathbf{R} := -\mathbf{A_0}(\mathbf{A_1} + \mathbf{A_0}\mathbf{G})^{-1}$
```

It can be seen that one iteration of the while-loop requires much more floating point operations than in the Successive Substitution method. However, the number of iterations is drastically reduced due to the quadratic convergence speed. Note that the required number of iterations depends on the distribution of the probability mass over the levels: the iteration stops when the state probabilities of the remaining levels above k (given by $||\mathbf{1} - \mathbf{G1}||_\infty$) can be neglected. As a consequence, queueing processes with a high average queue length require more iterations than those with a low average queue length.

Improved LR method

An improvement to the LR method has been presented in [82]. It reduces the computational effort per iteration of the original LR algorithm by avoiding some of the expensive matrix multiplications inside the iteration loop. For this purpose, the algorithm employs a factorization to the matrices $\mathbf{B_0}$ and $\mathbf{B_2}$. Additionally, the product $\mathbf{W} = \mathbf{A_0 G}$ is computed instead of $\mathbf{G}$. It is shown in [88] that these changes reduce the number of operations per iteration by about 24%. The pseudo code is given here (the matrices $\mathbf{M}, \mathbf{N}, \mathbf{L}, \mathbf{X}$ and $\mathbf{Y}$ have been introduced for the factorization):

```
$\mathbf{N} := \mathbf{A_1}, \mathbf{L} := \mathbf{A_0}, \mathbf{M} := \mathbf{A_2}$
$\mathbf{X} := -\mathbf{N}^{-1}\mathbf{L}, \mathbf{Y} := -\mathbf{N}^{-1}\mathbf{M}, \mathbf{Z} := \mathbf{LY}$
$\mathbf{W} := \mathbf{Z}$
while $||\mathbf{A_0 1} - \mathbf{W1}||_\infty \geq \varepsilon$
    $\mathbf{N} := \mathbf{N} + \mathbf{Z} + \mathbf{MX}, \mathbf{L} := \mathbf{LX}, \mathbf{M} := \mathbf{MY}$
    $\mathbf{X} := -\mathbf{N}^{-1}\mathbf{L}, \mathbf{Y} := \mathbf{N}^{-1}\mathbf{M}, \mathbf{Z} := \mathbf{LY}$
    $\mathbf{W} := \mathbf{W} + \mathbf{Z}$
end
$\mathbf{R} := -\mathbf{A_0}(\mathbf{A_1} + \mathbf{W})^{-1}$
```

3.3.4 Transform methods

Unlike the matrix-geometric solution methods the transform methods do not aim to directly solve Equation (3.11). Instead, they first transform the problem to some other domain in order to derive the solution of Equation (3.8). Three methods are presented here:

The Cyclic Reduction method

The Cyclic Reduction method [11] yields an efficient solution method for a class of processes that is more general than QBDs. For this purpose, results from the theory of block Toeplitz matrices are employed. Interestingly, this method is identical to the improved LR method (see above) when restricted to the class of QBD processes.

The Invariant Subspace method

The idea of the Invariant Subspace method [2] is to formulate the matrix-geometric solution matrix $\mathbf{R}$ in terms of a particular invariant subspace, namely the *left invariant subspace.* A detailed discussion of this approach is beyond this overview; we will only give some remarks about its performance.

The computation of the left invariant subspace is performed by evaluating the so-called *matrix sign function.* The developers of the Invariant Subspace method originally suggested a simple iterative procedure with quadratic convergence for this purpose. The number of operations per iteration is much higher than in the other approaches. However, it is *not* sensitive to the probability mass distribution, as opposed to the matrix-geometric solution methods.

The Spectral Expansion method

The Spectral Expansion method has been first proposed in [85] and thoroughly discussed in [20]. A good introduction can be found in [88]. Nevertheless, we will give a brief overview of the method because it yields a very special and handy presentation of the solution vector $\mathbf{v}$.

The Spectral Expansion approach states that the sub-vectors $\mathbf{v_i}$ ($i \geq 1$) are given by

$$\mathbf{v_i} = \boldsymbol{\psi}\lambda^{i-1}, \qquad i \geq 1, \tag{3.14}$$

where $\boldsymbol{\psi}$ and λ are the eigenvector resp. eigenvalue of the quadratic eigenvalue problem

$$\boldsymbol{\psi}(\mathbf{A_0} + \lambda\mathbf{A_1} + \lambda^2\mathbf{A_2}) = \mathbf{0}. \tag{3.15}$$

Note that only eigenvectors with corresponding eigenvalue $|\lambda| < 1$ yield valid solutions since otherwise the normalizing condition (3.5) is not satisfied. In general, more then one eigenvalue fulfilling $|\lambda| < 1$ exists. We denote these eigenvalues and their corresponding eigenvectors $\lambda_1, \ldots, \lambda_c$, resp. $\boldsymbol{\psi_1}, \ldots, \boldsymbol{\psi_c}$. The overall solution is then given by the linear combination

$$\mathbf{v_i} = \sum_{j=1}^{c} x_j \boldsymbol{\psi_j} \lambda_j^{i-1}, \qquad i \geq 1, \tag{3.16}$$

where x_j are the coefficients of the linear combination. The coefficients x_j as well as the boundary solution vector $\mathbf{v_0}$ can be computed using Equations (3.6), (3.7) and (3.9). The representation of the solution subvector $\mathbf{v_i}$ as power of the eigenvalues (see Equation (3.16)) allows to access any subvector without the expensive computation of the matrix-power in Equation (3.10).

Note that a naive implementation of the Spectral Expansion method would result in an algorithm that is much slower than the matrix-geometric algorithms, especially because the computations have to be done in the complex number space. Fortunately, it is possible to substantially optimize the computation by making use of the fact that all the eigenvalues and eigenvectors are either real or appear in complex conjugate pairs [88].

3.4 Finite QBDs

3.4.1 Definition

Similar to infinite QBDs, finite QBDs can be seen as the generalization of the queueing process of a bounded M|M|1|K queue. Finite QBD processes result in QBD Markov chains with a finite number $K+1$ of levels, hence two boundary levels can be identified: the *lower boundary level* 0 and the *upper boundary level* K.

In the following we will only treat a quite restricted class of finite QBDs that is sufficient for the queueing process discussed in this thesis: The upper boundary level has the same number of states N as the repeating levels 1 through $K-1$. Additionally, the transition rates between level $K-1$ and K are the same as between the repeating levels — only one new matrix $\mathbf{C}$ is introduced that specifies the transition rates inside level K. The finite generator matrix of the QBD Markov chain then has the following form:

$$\mathbf{Q} = \begin{pmatrix} \mathbf{B_{0,0}} & \mathbf{B_{0,1}} & & & & & \\ \mathbf{B_{1,0}} & \mathbf{A_1} & \mathbf{A_0} & & & & \\ & \mathbf{A_2} & \mathbf{A_1} & \mathbf{A_0} & & & \\ & & \mathbf{A_2} & \mathbf{A_1} & \mathbf{A_0} & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & \mathbf{A_2} & \mathbf{A_1} & \mathbf{A_0} \\ & & & & & \mathbf{A_2} & \mathbf{C} \end{pmatrix}. \tag{3.17}$$

3.4.2 Steady-state solution

We search the finite steady-state probability vector $\mathbf{v}$ of the QBD Markov chain with generator matrix $\mathbf{Q}$ that fulfills the global balance equation

$$\mathbf{v} \cdot \mathbf{Q} = \mathbf{0} \tag{3.18}$$

and the normalization condition

$$\mathbf{v} \cdot \mathbf{1} = 1. \tag{3.19}$$

As in the infinite case, we partition the vector $\mathbf{v}$ into subvectors $\mathbf{v}_i$ where $\mathbf{v_0} = (v_1, \ldots, v_{N_0})$ and $\mathbf{v_i} = (v_{N_0+1+(i-1)\cdot N}, \ldots, v_{N_0+i\cdot N})$, for $i = 1, \ldots, K$. It is important to note that the solution of Equation (3.18) is uncritical with respect to space complexity. Due to the special structure of the Markov chain it is not necessary to hold the whole matrix $\mathbf{Q}$ in memory but only the matrices $\mathbf{B_{0,0}}$, $\mathbf{B_{1,0}}$, $\mathbf{B_{0,1}}$, $\mathbf{B_{1,1}}$, $\mathbf{A_0}$, $\mathbf{A_1}$, $\mathbf{A_2}$ and $\mathbf{C}$. In terms of these matrices, Equation (3.18) becomes:

$$\begin{aligned}
\mathbf{v_0 B_{0,0}} + \mathbf{v_1 B_{1,0}} &= \mathbf{0} && (3.20)\\
\mathbf{v_0 B_{0,1}} + \mathbf{v_1 B_{1,1}} + \mathbf{v_2 A_2} &= \mathbf{0} && (3.21)\\
\mathbf{v_i A_0} + \mathbf{v_{i+1} A_1} + \mathbf{v_{i+2} A_2} &= \mathbf{0}, \qquad \text{for } i = 1, \dots, K-2, && (3.22)\\
\mathbf{v_{K-1} A_0} + \mathbf{v_K C} &= \mathbf{0} && (3.23)
\end{aligned}$$

Since $\mathbf{Q}$ is of finite size, Equation (3.18) can be solved by an ordinary Gauss-Seidel-iteration which performs very efficiently due the band-structure of $\mathbf{Q}$. More sophisticated algorithms have been developed on the basis of the solution methods for infinite QBDs; most of the algorithms presented in Section 3.3 have been extended to the treatment of finite QBDs.

In addition to these methods some authors have developed solution methods especially adapted to QBDs arising from PH|PH|1|K queues (see Section 5.3). In the following we will present four of them. We assume that the arrival process and the service process are specified by $(\boldsymbol{\alpha}, \mathbf{A})$ resp. $(\boldsymbol{\beta}, \mathbf{B})$. The vectors $\mathbf{A^0}$ and $\mathbf{B^0}$ are defined by $\mathbf{A^0} = -\mathbf{A} \cdot \mathbf{1}$, $\mathbf{B^0} = -\mathbf{B} \cdot \mathbf{1}$.

3.4.3 Solution algorithms

The Bocharov-Naoumov method

The method proposed in [14] is a matrix-geometric solution method. It first introduces the matrices $\mathbf{W_0}$, $\mathbf{W}$, $\mathbf{W_{K-1}}$ and $\mathbf{V}$ with

$$\begin{aligned}
\mathbf{W_0} &= -(\mathbf{A} \otimes \beta)\tilde{\mathbf{B}}^{-1}, && (3.24)\\
\mathbf{W} &= \tilde{\mathbf{A}}\tilde{\mathbf{B}}^{-1}, && (3.25)\\
\mathbf{W_{K-1}} &= (\mathbf{A^0}\boldsymbol{\alpha} \otimes \mathbf{I})\left[(\mathbf{A} + \mathbf{A^0}\boldsymbol{\alpha}) \oplus \mathbf{B}\right]^{-1}, && (3.26)\\
\mathbf{V} &= \mathbf{I} + \left(\sum_{i=0}^{K-2} \mathbf{W_0 W^i} + \mathbf{W_0 W^{K-2} W_{K-1}}\right)(\mathbf{I} \otimes \mathbf{1}), && (3.27)
\end{aligned}$$

where $\tilde{\mathbf{A}} = (\mathbf{1}\boldsymbol{\alpha} - \mathbf{I}) \otimes \mathbf{B} - \mathbf{A} \otimes \mathbf{I}$, $\tilde{\mathbf{B}} = \mathbf{A} \otimes (\mathbf{1}\boldsymbol{\beta} - \mathbf{I}) - \mathbf{I} \otimes \mathbf{B}$. Let $\mathbf{u}$ be the solution of the system of equations

$$\begin{aligned}
\mathbf{u}(\mathbf{A} + \mathbf{A^0}\boldsymbol{\alpha}) &= \mathbf{0}, && (3.28)\\
\mathbf{u1} &= 1. && (3.29)
\end{aligned}$$

Then, the steady-state probability subvector $\mathbf{v_0}$ can be obtained by solving

$$\mathbf{v_0 V} = \mathbf{u}. \qquad (3.30)$$

The other subvectors $\mathbf{v_i}$ $(i = 1, \ldots, K)$ are given by

$$\mathbf{v_i} = \begin{cases} \mathbf{v_0 W_0 W}^{i-1}, & i = 1, \ldots, K-1, \\ \mathbf{v_0 W_0 W}^{K-2}\mathbf{W_{K-1}}, & i = K. \end{cases} \tag{3.31}$$

Unfortunately, our experiments have shown that this approach suffers from considerable numerical instabilities. After a few iterations, very small as well as very large numbers occur in the computation of the subvectors $\mathbf{v}_i$ and, as a consequence, invalid results are returned.

The Chakravarthy-Neuts algorithm 1

This algorithm has been presented in [21] and follows the QBD approach. Instead of computing the steady-state probability vector $\mathbf{v}$ directly, it proposes a Gauss-Seidel iteration for the vectors $\mathbf{x}$ and $\mathbf{z}$, where $\mathbf{x}$ is the steady-state probability vector of the Markov chain embedded at points of departure, and $\mathbf{z}$ is the steady-state probability vector of the Markov chain embedded at points of arrival. Once $\mathbf{x}$ and $\mathbf{z}$ are known, $\mathbf{v}$ can be computed using the following equations:

$$k'\mathbf{z_0} = \mathbf{v_0 A^0} = k\mathbf{x_0 1}, \tag{3.32}$$
$$k'\mathbf{z_i} = \mathbf{v_i}(\mathbf{A^0} \otimes \mathbf{I}), \qquad i = 1, \ldots, K, \tag{3.33}$$
$$k\mathbf{x_{i-1}} = \mathbf{v_i}(\mathbf{I} \otimes \mathbf{B^0}), \qquad i = 1, \ldots, K, \tag{3.34}$$

where k and k' are normalizing constants and $\mathbf{x_i}$ $(i = 0, \ldots, K-1)$ and $\mathbf{z_i}$ $(i = 0, \ldots, K)$ are defined as the level subvectors of $\mathbf{x}$ resp. $\mathbf{z}$. The authors introduce the matrices $\mathbf{C_{11}}$, $\mathbf{C_{12}}$, $\mathbf{C_{21}}$ and $\mathbf{C_{22}}$ given by

$$\begin{aligned} \mathbf{C_{11}} &= -(\boldsymbol{\alpha} \otimes \mathbf{I})(\mathbf{A} \oplus \mathbf{B})^{-1}(\mathbf{A^0} \otimes \mathbf{I}), & \mathbf{C_{12}} &= -(\boldsymbol{\alpha} \otimes \mathbf{I})(\mathbf{A} \oplus \mathbf{B})^{-1}(\mathbf{I} \otimes \mathbf{B^0}), \\ \mathbf{C_{21}} &= -(\mathbf{I} \otimes \boldsymbol{\beta})(\mathbf{A} \oplus \mathbf{B})^{-1}(\mathbf{A^0} \otimes \mathbf{I}), & \mathbf{C_{22}} &= -(\mathbf{I} \otimes \boldsymbol{\beta})(\mathbf{A} \oplus \mathbf{B})^{-1}(\mathbf{I} \otimes \mathbf{B^0}). \end{aligned} \tag{3.35}$$

where $\oplus$ is the Kronecker sum. Equations (3.32)–(3.34) lead then to a recursive scheme for $\mathbf{z}$:

$$k'\mathbf{z_0} = k\mathbf{x_0 1}, \tag{3.36}$$
$$k'\mathbf{z_1} = (k'\mathbf{z_0}\alpha + k\mathbf{x_1})\mathbf{C_{21}}, \tag{3.37}$$
$$k'\mathbf{z_i} = k'\mathbf{z_{i-1}C_{11}} + k\mathbf{x_i C_{21}}, \qquad i = 2, \ldots, K-1, \tag{3.38}$$
$$k'\mathbf{z_k} = k'(\mathbf{z_{k-1}} + \mathbf{z_k})\mathbf{C_{11}}. \tag{3.39}$$

Similar relations can be derived for $\mathbf{x}$. Based on these equations the authors propose a Gauss-Seidel iteration for $\mathbf{x}$ and $\mathbf{z}$. We will not reprint its pseudo-code here but it is interesting how $\mathbf{x}$ and $\mathbf{z}$ are initialized for the iteration: the authors propose that a convenient initial solution for $\mathbf{x}$ can be obtained from the steady-state probability vector of the PH|M|1|K queue with the same arrival process and same service rate as the original PH|PH|1|K queue. Similarly, the initial solution of $\mathbf{z}$ is drawn from the M|PH|1|K model with the same arrival rate and same service process as the original queue. For both of these latter types of queues closed-form solutions are available.

The Chakravarthy-Neuts algorithm 2

This algorithm has also been proposed in [21]. It derives the solution vector $\mathbf{v}$ from the steady-state probability vector of the embedded Markov renewal process (MRP) at epochs of departure. Although the final algorithm is quite compact, its derivation would fill several pages, hence we will only give a statement about its performance: the authors state that while the MRP approach requires more overhead computations than the Chakravarthy-Neuts algorithm 1 (see above) it involves manipulating matrices of lower dimension. The results in [21] show that the MRP approach is up to three times faster than the QBD approach.

The Cyclic-Reduction method for PH|PH|1|K queues

The generator matrix $\mathbf{Q}$ of a PH|PH|1|K queueing process has an outer block structure (block tridiagonal block Toeplitz-like) and an inner structure given by the Kronecker sums and products. Some algorithms (like Gauss-Seidel iteration) exploit the inner structure but not fully the outer structure, other algorithms (like cyclic reduction) exploit the outer structure but may destroy the inner structure. The algorithm in [10] exploits the double structure of $\mathbf{Q}$ and is based on cyclic reduction where the iteration formulae are expressed in a form that allows to take advantage of the inner structure. The algorithm consists of two parts where the first part only depends on the arrival and service processes and not on the queueing capacity.

3.5 Performance of the solution methods

First, we notice that all solution methods are uncritical with respect to space complexity because only the block matrices $\mathbf{B_{0,0}}, \mathbf{B_{0,1}}, \ldots$ and some temporary data structures of small size have to be held in main memory. The time complexity of the methods is briefly discussed in the following.

Infinite QBDs

For infinite QBDs, [88] gives a detailed analysis of the time complexity of the solution methods. Table 3.1 summarizes these time complexities (measured in number of floating-point operations) for solving the boundary level of size N_0 and the repeating levels of size N of QBD systems. The right column gives the typical number

method	boundary	repeating levels	typical
Succ. Substitution	$\frac{1}{3}(N_0+N)^3+\frac{4}{3}N^3$	$\frac{1}{3}N^3+n\cdot\frac{9}{3}N^3$	$n \gg 100$
LR	$\frac{1}{3}(N_0+N)^3+\frac{4}{3}N^3$	$\frac{14}{3}N^3+n\cdot\frac{25}{3}N^3$	$n < 40$
Improved LR	$\frac{1}{3}(N_0+N)^3+\frac{4}{3}N^3$	$\frac{16}{3}N^3+n\cdot\frac{19}{3}N^3$	$n < 40$
Invariant Subspace	$\frac{1}{3}(N_0+N)^3+\frac{4}{3}N^3$	$\frac{28}{3}N^3+n\cdot\frac{24}{3}N^3$	$n < 40$
Spectral Expansion	$\frac{1}{3}(N_0+N)^3+\frac{6}{3}N^3+N^2N_0$	$\frac{7}{3}N^3+\frac{639}{3}N^3$	–

Table 3.1: Time complexities of algorithms for infinite QBDs (from [88])

of iterations n that a solution method requires to reach a given accuracy. For the matrix-geometric solution methods, n is sensitive to the probability mass distribution. Experiments in [88] with a PH|PH|1 queueing process showed that the computation time of the (improved) LR method tripled when the traffic intensity was increased from 0.1 to 0.97 whereas the Spectral Expansion and the Invariant Subspace method were nearly insensitive to changes of the traffic intensity. On a 233 MHz x86 CPU, the slowest algorithm (the Invariant Subspace approach) took about 50 seconds for $N_0 = N = 161$ with IEEE double precision. The other methods took about 10 through 30 seconds, depending on the traffic density.

Regarding the solution quality of the algorithms, [88] reports that the accuracies achieved by the Spectral Expansion algorithm for the boundary solution vector significantly depend on the traffic intensity and the repeating level size N. It is recommended to restrict the Spectral Expansion algorithm to QBDs with $N < 40$, especially under high traffic densities.

Finite QBDs

As already stated, Equation (3.18) can be solved by an ordinary Gauss-Seidel iteration. Due to the band structure of the generator matrix $\mathbf{Q}$, the number of non-zero elements in the generator matrix only depends linearly on the upper boundary level K: level 0 contains $N_0^2 + 2N_0N$ (non-zero) elements, level 1 through $K-1$ contain $3N^2$ elements each, and level K contains N^2 elements. In total, the Gauss-Seidel method has to process

$$N_0^2 + 2N_0N + 3N^2(K-1) + N^2 \approx 3KN^2$$

elements per iteration. We have observed in experiments with PH|PH|1|K queues that the queueing capacity K enters with about $K^{\frac{3}{2}}$ into the number of iterations when the offered load is in the range from 95% to 105%. In some sample queueing stations with high load and $K = 200$ this resulted in up to 4500 iterations.

Other methods can be considerably faster: for example, the Cyclic Reduction method for PH|PH|1|K queues has a total numerical complexity of $O((l+m)^3 \log K + (l+m)^2 K)$ where l and m are the number of phases of the arrival, resp. service PH distribution (hence $N = l \cdot m$). The authors of [10] report a speed-up in comparison to Gauss-Seidel that increases as K grows, varying between 168 for $K = 1600$ with $N = 25$ and 382 for $K = 1600$ with $N = 10$.

3.6 Summary and conclusions

In this chapter we have introduced the fundamental mathematical structures and notations that are used in the following chapters. We have described the Markovian arrival processes which cover many interesting point processes, including the Markov-modulated Poisson processes and the phase-type renewal processes.

The queueing processes that we will discuss in the next chapters have underlying Markov chains that belong to the well-known class of continuous-time Quasi-Birth-and-Death processes. We have given the formal definition of infinite QBDs and finite QBDs, as well as various solution methods to compute their steady state solution. All solution methods have a low space complexity. On today's hardware, this fact allows most algorithms to run nearly entirely from the L2 cache of the CPU for level sizes $N, N_0 < 100$. For infinite QBDs, the improved LR algorithm and the Spectral Expansion approach are the fastest methods we are aware of. The Spectral Expansion method is of interest because it is insensitive to extreme traffic rates, but should be, due to the accuracy issues, only used for QBDs with $N < 40$. Additionally, the accuracy of the (improved) LR approach can be easily controlled a priori, which is not possible for the eigenvalue computation of the Spectral Expansion method. In case of finite QBDs, a simple Gauss-Seidel iteration is often sufficient for QBDs of medium size ($K \leq 150$ and $N \leq 30$). The availability of better algorithms, like the Cyclic Reduction, allows to analyze QBDs of considerably larger size in nearly real-time.

The numerical results presented in the next chapters have been computed by the LR algorithm for infinite QBDs and by a Gauss-Seidel iteration or the Cyclic Reduction method for finite QBDs.

Chapter 4

Fitting of long-tailed traffic traces to hyper-exponential distributions

Over the last decade, extensive traffic measurements have shown the presence of properties such as *self-similarity*, *fractality* and *long-range dependency* in network traffic. The seminal paper by Leland *et al.* [69] showed self-similarity in Ethernet traffic; later, similar effects were shown to exist in wide area network traffic, signaling traffic, and in multimedia and video traffic. Crovella and Bestavros [24] have shown that network traffic that is due to WWW transfers can show characteristics that are consistent with self-similarity and argue that this can be explained by the heavy-tailedness of many of the involved distributions. Ignoring the above properties (that is, self-similarity, heavy-tailedness) in the analysis of queueing systems leads, in general, to an under-valuation of important performance measures [93].

Various efforts have been pursued to develop appropriate traffic models to evaluate the performance of systems under self-similar traffic. Often, the first step is to construct heavy-tailed distributions (HTDs) to approximate the involved empirical distributions in measurement data. However, "classical" HTDs cannot be used so easily for analytical or numerical evaluation studies, since the latter often rely on the use of Poisson or other "Markovian" distributions. To overcome this problem, various approaches to approximate HTDs by analytically more tractable distributions have been proposed [16, 30, 32, 34, 37, 90, 96, 97, 110].

Of particular interest is the use of hyper-exponential distributions (HEDs) to approximate HTDs, since these distributions are very well understood and well-suited for analytical and numerical performance studies. Indeed, an HED is an example of a phase-type distribution (see Section 3.2). The use of HEDs for this purpose has been proposed by, among others, Feldmann and Whitt [34] (denoted here as the "FW-approach"). Although the FW-approach is fast, it requires an explicit representation of an HTD to fit to. Such an explicit HTD can for instance be obtained by fitting a Weibull or a Pareto distribution to the measurement data. However, as we will see below, often the measurements to be fitted do not suit a Weibull or a Pareto distribution well, so that the thus obtained HED does not describe the measurements well.

To avoid the use of an intermediate HTD, we have proposed in [29, 31] to directly fit an HED to the measured data via the Expectation Maximization method (EM) [27]. This approach has also been followed by others [90, 96, 97, 110].

This chapter is further organized as follows. We will give some background on HTDs and HEDs in Section 4.1. The FW-approach is summarized in Section 4.2. The EM-algorithm and its specialization to HEDs are discussed in Section 4.3. In Section 4.4, we describe two EM-based fitting methods by other authors that increase the efficiency and the accuracy of the EM-algorithm. In Section 4.5 we present an EM-based algorithm that applies a sampling and stratification method to the data in order to increase the efficiency of the fitting. Our approaches are validated in Section 4.6. The chapter is concluded in Section 4.7. Note that Sections 4.1 through 4.3 and parts of Section 4.6 are based on [29, 31] and have also been presented in [28].

4.1 Heavy-tailed distributions and hyper-exponential distributions

In this section, we give some background on heavy-tailed distributions in Section 4.1.1 and hyper-exponential distributions in Section 4.1.2.

4.1.1 Heavy-tailed distributions

Self-similarity in network traffic has been explained by the fact that many of the involved distributions, e.g., of file sizes, are heavy-tailed. In an HTD, the complementary cumulative distribution function F^c decays more slowly than exponentially, i.e., $e^{\theta x}F^c(x) \to \infty$ as $x \to \infty$ for all $\theta > 0$. For a random variable X, distributed according to some HTD, we typically have:

$$P[X > x] \sim x^{-\alpha}, \qquad x \to \infty, \qquad 0 < \alpha < 2.$$

Note that "$x \to \infty$" should be read as "for very large x" in case of measurements.

The degree of the heavy-tailedness is given by the value of the shape parameter α which can be determined by plotting the complementary cumulative distribution $F^c(x) = 1 - F(x) = P[X > x]$ on a log-log scale. The slope of the plot, found, for instance, via a linear regression, then gives the value of α.

In Table 4.1, we list some characteristics of two well-known HTDs [55], the Pareto and the Weibull distribution (in case the stated conditions are not met, the expectation and/or variance do not exist).

distribution	density $f(x)$	expectation	variance
Pareto	$ak^a x^{-(a+1)}$	$\frac{ak}{a-1}$, for $a > 1$	$\frac{ak^2}{(a-2)(a-1)^2}$, for $a > 2$
Weibull	$\frac{b}{a^b} x^{b-1} e^{-(x/a)^b}$, for $a > 0$ and $b > 0$	$\frac{a}{b}\Gamma(1/b)$	$\frac{a^2}{b^2}\{2b\Gamma(2/b) - [\Gamma(1/b)]^2\}$

Table 4.1: Characteristics of the Pareto and Weibull distribution

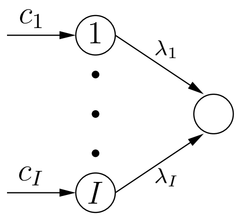


Figure 4.1: Graphical representation of an I-phase hyper-exponential distribution

4.1.2 Hyper-exponential distributions

An HED can be interpreted as a probabilistic choice between I exponential distributions (see Figure 4.1) and is an example of a phase-type distribution. With (initial) probability c_i the i-th negative exponential distribution (with rate λ_i) is chosen. Such an I-phase HED has distribution function

$$F(x) = 1 - \sum_{i=1}^{I} c_i e^{-\lambda_i x}, \tag{4.1}$$

and density

$$f(x) = \sum_{i=1}^{I} c_i \lambda_i e^{-\lambda_i x}.$$

Its j-th moment is given by

$$E[X^j] = j! \cdot \sum_{i=1}^{I} \frac{c_i}{\lambda_i^j}.$$

Note that, for $I \to \infty$, one can represent *any* distribution with squared coefficient of variation at least 1 and with completely monotone probability density function arbitrary close by hyper-exponentials [34]. However, it has been shown that with values of I up to 20 [34], HTDs can approximate Weibull and Pareto distributions for large ranges of x.

4.2 Approximation of HTDs with HEDs

The approach by Feldmann and Whitt (FW) [34] comprises an efficient and elegant method to approximate an HTD with an HED. The method is often applied due to its simplicity and efficiency [37, 52]. In this section we first summarize the approach before we discuss our experience with it.

The FW-approach

In the FW-approach, it is assumed that an HTD is given in an explicit form. How this HTD is obtained from, for instance, measurement data, is not a part of it. Provided that an explicit representation of the HTD $F(x)$ is available, an I-phase HED of the form given in Equation (4.1) is found.

For a given HTD $F(x)$ and an *a priori* fixed number of phases I, the FW-approach operates as follows:

1. Choose quantiles $0 < q_I < q_{I-1} < \ldots < q_1$ with sufficiently large ratio q_i/q_{i+1}, e.g., $q_i/q_{i+1} \approx 10$ (for $i = 1, \ldots, I-1$). Furthermore, let b be such that $1 < b < q_i/q_{i+1}$ for all i.

2. In I steps, the parameters for the phases in the HED are computed. We start with setting $i := 1$ and $F_i^c(x) = F_1^c(x) = 1 - F(x)$.

3. In the i-th phase, we compute c_i and λ_i by solving the equations

$$\begin{aligned} c_i e^{-\lambda_i q_i} &= F_i^c(q_i), \\ c_i e^{-\lambda_i b q_i} &= F_i^c(bq_i), \end{aligned}$$

   yielding (explicitly)

$$\lambda_i = \frac{1}{(b-1)q_i} \ln\left(\frac{F_i^c(q_i)}{F_i^c(bq_i)}\right) \quad \text{and} \quad c_i = F_i^c(q_i) e^{\lambda_i q_i}.$$

4. Step 3 is repeated for $i = 2, \ldots, I-1$ where

$$\begin{aligned} F_i^c(q_i) &= F_{i-1}^c(q_i) - c_{i-1} e^{-\lambda_{i-1} q_i}, \\ F_i^c(bq_i) &= F_{i-1}^c(bq_i) - c_{i-1} e^{-\lambda_{i-1} b q_i}. \end{aligned}$$

5. Finally, for the last phase I we find $c_I := 1 - \sum_{j=1}^{I-1} c_j$, and λ_I follows from $c_I e^{-\lambda_I q_I} = F_I^c(q_I)$.

The complexity of the algorithm is $O(I)$ where each step consists of solving a system of two, in fact, linear equations. However, since the algorithm cannot be applied directly to measurement data, the costs of an algorithm, like the ML-algorithm [27], to fit the measurements to an explicit HTD must be considered as well.

Application and validation

When applying the FW-approach to find object-size distributions from the log-files used in our case studies (for a detailed description of the traces and the statistical parameters, see Section 4.6), we found that the typically employed HTDs, like Pareto and Weibull, do not describe the object-size distributions well. Both distributions fit the tail of empirical distributions well, but fail to fit the head and waist properly. For example, a Weibull distribution whose first and second moment have been fitted to the data, results in a median that is half the median found in the data (the median is located in the head; see Table 4.3 and Table 4.4 in Section 4.6).

Hence, even when the FW-approach does give a good fit with respect to a given HTD, if the provided HTD does not describe the data well, then the finally fitted HED does not describe the measurements well, either.

Feldmann and Whitt [34] point out that it might be possible to extend their approach so it can be directly applied to measurement data. They also warned that the algorithm, at least without extension, is not designed to directly treat data but might well be applied after some initial smoothing of the data. We have performed a number of experiments in this direction. In fact, these experiments have shown that the smoothing is absolutely necessary, since otherwise the algorithm is too sensitive to the location of the quantiles q_i. Furthermore, the quality of the approximation heavily depends on the quality of the smoothing. Simple smoothing methods based on linear or square interpolation did not yield satisfactory results.

4.3 EM-fitting with HEDs

The Expectation Maximization method (EM) is a well-known algorithm to fit measurements to distributions [6, 25, 99, 105]. The EM-algorithm operates in an iterative fashion and does require neither an intermediate HTD nor any heuristics. Below, we outline the method in general in Section 4.3.1, and then specialize it in Section 4.3.2 to the case where the distribution function to fit to is an HED [29, 31]. Its complexity and the choice of the initial values are discussed in Section 4.3.3.

4.3.1 General approach

Given measurement data $x_1, \ldots, x_N$, we search the parameters $\mathbf{c} = (c_1, \ldots, c_I)$ and $\boldsymbol{\theta} = (\boldsymbol{\theta_1}, \ldots, \boldsymbol{\theta_I})$ of the density function

$$p(x|(\mathbf{c}, \boldsymbol{\theta})) = \sum_{i=1}^{I} c_i \cdot p(x|\boldsymbol{\theta_i}), \tag{4.2}$$

so that it "best" fits the density of the measurement data. The density in Equation (4.2) is a convex combination of basic density functions $p(x|\boldsymbol{\theta_i})$ parameterized by

$\boldsymbol{\theta}_\mathbf{i}$ with weights $c_i \geq 0$ and $\sum_{i=1}^{I} c_i = 1$. Now, let $\alpha = (\mathbf{c}, \boldsymbol{\theta})$ and $\alpha' = (\mathbf{c'}, \boldsymbol{\theta'})$ be two sets of parameters for the density p. The EM-algorithm defines a new probability mass function

$$\delta(i|x_n, \alpha) = \frac{c_i \cdot p(x_n|\boldsymbol{\theta}_\mathbf{i})}{p(x_n|\alpha)},$$

as well as the function

$$Q(\alpha, \alpha') = \sum_{n=1}^{N} \sum_{i=1}^{I} \delta(i|x_n, \alpha) \cdot \log(c'_i \cdot p(x_n|\boldsymbol{\theta'}_\mathbf{i})),$$

which provides a quality criterion for α and α': it says how much better the density function $p(x|\alpha')$ fits the measurement data than the density function $p(x|\alpha)$.

The EM algorithm proceeds iteratively: starting from an initial parameter set α, it computes a new parameter set α' such that $Q(\alpha, \alpha')$ is maximized. This α' is used as starting point for the next iteration. The algorithm stops when $\alpha \approx \alpha'$ (see below). To find the next value α', that is, to optimize Q, one has to take derivatives to subsequently solve the (possibly non-linear) equation system:

$$\frac{\partial Q}{\partial \alpha'} = 0 \Rightarrow \frac{\partial Q}{\partial \boldsymbol{\theta'}_\mathbf{1}} = 0, \cdots, \frac{\partial Q}{\partial \boldsymbol{\theta'}_\mathbf{I}} = 0. \tag{4.3}$$

Using Lagrange multipliers (with auxiliary condition $\sum_{i=1}^{I} c_i = 1$), the new weights are given by:

$$c'_i = \frac{1}{N} \sum_{n=1}^{N} \delta(i|x_n, \alpha).$$

In general, the equation system (4.3) is difficult to solve. However, in case we take hyper-exponentials as basic densities, this becomes easily feasible. We discuss this in the next section.

4.3.2 Specialization to HEDs

We now take HEDs as basic densities, i.e., $p(x|\lambda_i) = \lambda_i e^{-\lambda_i x}$. Equation (4.3) yields

$$\frac{\partial Q}{\partial \lambda'_i} = 0 \quad \Rightarrow \quad \sum_{n=1}^{N} \delta(i|x_n, \alpha) \cdot \frac{\partial}{\partial \lambda'_i} \log\left(c'_i \cdot p(x_n|\lambda'_i)\right) = 0.$$

Substituting the function $p(x_n|\lambda'_i)$ gives us

$$\sum_{n=1}^{N} \delta(i|x_n, \alpha) \cdot \frac{\partial}{\partial \lambda'_i} \log\left(c'_i \cdot \lambda'_i \cdot e^{-\lambda'_i x_n}\right) = 0,$$

$$\Rightarrow \sum_{n=1}^{N} \delta(i|x_n, \alpha) \cdot \frac{\partial}{\partial \lambda'_i} \left(\log c'_i + \log \lambda'_i - \lambda'_i x_n\right) = 0,$$

1. Select an appropriate number of distributions I and initial parameters $\alpha = (c_1, \ldots, c_I, \lambda_1, \ldots, \lambda_I)$, as well as a positive required accuracy ε.
2. Compute for $i := 1$ to I:
   (a) $\delta(i|x_n, \alpha) = \frac{c_i \cdot p(x_n|\lambda_i)}{p(x_n|\alpha)}$,
   (b) $c_i' = \frac{1}{N} \sum_{n=1}^{N} \delta(i|x_n, \alpha)$,
   (c) $\lambda_i' = \frac{\sum_{n=1}^{N} \delta(i|x_n,\alpha)}{\sum_{n=1}^{N} \delta(i|x_n,\alpha) \cdot x_n}$.
3. Return to step 2 with $c_i := c_i'$ and $\lambda_i := \lambda_i'$ until the difference between c_i and c_i' and/or the difference between λ_i and λ_i', for all i, is smaller than the accuracy boundary ε.

Figure 4.2: EM-algorithm for HEDs

$$\Rightarrow \sum_{n=1}^{N} \delta(i|x_n, \alpha) \cdot (1/\lambda_i' - x_n) = 0,$$

$$\Rightarrow \frac{\sum_{n=1}^{N} \delta(i|x_n, \alpha)}{\lambda_i'} = \sum_{n=1}^{N} \delta(i|x_n, \alpha) \cdot x_n,$$

$$\Rightarrow \lambda_i' = \frac{\sum_{n=1}^{N} \delta(i|x_n, \alpha)}{\sum_{n=1}^{N} \delta(i|x_n, \alpha) \cdot x_n}.$$

The EM-algorithm specialized for HEDs now takes the form shown in Figure 4.2. Note that we have preset the number of phases I for the algorithm. In a different variant of the EM-algorithm, this number does not have to be preset, but is computed on-the-fly, thus yielding a number of phases that is large enough to describe the required HTD, yet as small as possible to keep the fitted HED small [25, 72].

4.3.3 Complexity and choice of initial values

The EM-algorithm is an iterative algorithm where the complexity of each iteration is $O(N \cdot I)$, with N the number of measurement samples and I the number of phases. A problem with the EM-algorithm is the fact that it is difficult to predict the number of iterations needed to reach a given precision of the result [6]. However, in our experiments, good results generally have been obtained within $10 - 30$ iterations. Additionally, it should be noted that even for a case study with N well over 17 million (see below) and $I = 5$, one iteration takes less than 10 seconds on a standard personal computer.

The number of required iterations is heavily influenced by the choice of the initial values. Here, we can use our knowledge about the shape of the distribution function to choose initial values that are near to the (expected) final results of the algorithm. We know, e.g., that data sizes in the world-wide web are described by HTDs, and that small documents are more popular than larger ones. That means the values for c_i have to be decreasing according to the values of the data sizes, given by the inverses of λ_i. This results in the following guidelines for setting initial values for c_i and λ_i (for $i = 1, \ldots, I$):

$$c_i = \frac{9 \cdot 10^{-i}}{1 - 10^{-I}} \quad \text{and} \quad \lambda_i = 10^{-2-i}, \qquad \text{for } i = 1, \ldots, I. \tag{4.4}$$

When the number of phases I is not preset but computed on-the-fly (as mentioned at the end of Section 4.3.2), one starts with $I = 1$, $c_1 = 1$ and $\lambda_1 = N/(\sum_{n=1}^{N} x_n)$, that is, one starts with an expression according to ML-fitting. The algorithm then chooses a new number of phases $I_{new} := 2 \cdot I$, that is, the number of phases is *doubled* per iteration step. We refer to [29, 31] for further details on this.

4.4 Enhancements of the EM-fitting for traffic traces

In this section, we describe two EM-based fitting methods (developed recently by other authors) that increase the efficiency and the accuracy of our EM-algorithm. The Divide-and-Conquer-EM algorithm is presented in Section 4.4.1. An EM-algorithm based on the aggregation of measurement data is presented in Section 4.4.2.

4.4.1 Divide-and-Conquer-EM

The Divide-and-Conquer-EM algorithm (D&C-EM) [96] does not directly operate on the trace data. Instead, the data is first partitioned and the EM-algorithm is then used to fit an HED to each partition. The thus obtained HEDs are then composed to a final HED for the overall trace. In this way, the accuracy is increased because the approach reduces the possibility that the EM-algorithm does not correctly capture parts of the distribution, e.g., the heavy-tail, while searching for the global optimal solution.

To increase the efficiency, the partitions are selected so that they exhibit a lower variability than the variability of the entire trace. This leads to a faster convergence of the EM-algorithm. The coefficient of variation is used to determine the partition bounds. Starting from the most frequent value (according to the histogram of the data trace), values are accumulated into a partition until the coefficient of variation for that partition is larger than a given threshold c_{max} (between 1.2 and 1.5). When the threshold is reached, a new partition is created. The complete algorithm is outlined in the following:

1. Build the continuous data histogram of the trace data.

2. For each partition, accumulate data until the accumulated coefficient of variation for that partition is larger than a threshold c_{max}.

3. Apply EM (according to Section 4.3) to each partition.

4. Compose the obtained distributions to the final result.

The D&C-EM algorithm presented in [96] fits the data to an HED. Speed-ups of 10 to 1000 in comparison to the original EM-algorithm, depending on the number of phases and the characteristics of the data set, have been reported there. In [97], the authors extend the approach to a mixture of an Erlang and a hyper-exponential distribution which allows to approximate data sets with non-monotonically decreasing probability density function. The Erlang distribution with two phases is used to fit the first 0.5% distribution quantile and the hyper-exponential distribution models the rest of the density function.

4.4.2 Phase-type fitting with aggregated traffic traces

In [110, 111], an EM-algorithm, denoted as G-FIT, has been presented for fitting mixtures of Erlang distributions, so-called hyper-Erlang distributions, to trace data. A hyper-Erlang distribution (HErD) [33] is a mixture of I mutually independent Erlang distributions with order $r_1, \ldots, r_I$, rates $\lambda_1, \ldots, \lambda_I$, and initial probabilities $c_1, \ldots, c_I$. Obviously, the hyper-exponential distribution described in Section 4.1.2 is a special case of the hyper-Erlang distribution where $r_i = 1$, $i = 1, \ldots, I$. It has been shown in [111] that any probability density functions of a non-negative random variable can be approximated sufficiently close by an HErD.

In [90], G-FIT is extended by an aggregation algorithm that aims to reduce the size of the data set before the EM-algorithm actually is applied. Two types of aggregation are distinguished. In so-called *uniform trace aggregation*, the range of possible data values is divided into N^* intervals $[\Delta_{i-1}, \Delta_i]$ of identical width. Then, for each interval $i = 1, \ldots, N^*$ a tuple $(\widehat{x}_i, w_i)$ is determined where $\widehat{x}_i$ is the mean of all trace values in the interval $[\Delta_{i-1}, \Delta_i]$ and w_i is the proportion of trace values in that interval. The result is an *aggregated trace*, a set of N^* tuples, which is much smaller than the original data set and which is used as input data for a modified version of G-FIT.

For heavy-tailed distributions, the uniform partitioning does not perform well. Due to the heavy-tailedness, the data set covers a large range of values. One would be forced to use a large number of intervals in order to well approximate the distribution. For this reason, the authors also propose a *logarithmic trace aggregation* where the intervals have equal width on the logarithmic scale.

The experiments in [90] show that the reduction of a heavy-tailed distributed data trace with more than 10^6 elements to an aggregate trace with some hundred elements yields accurate results, whereas the CPU time requirements of the overall EM-algorithm (all iterations) decrease to a few seconds.

4.5 EM-algorithm with stratification

We have seen that the complexity of the EM-algorithm directly depends on the number of observations to process per iteration. A common method to reduce this number is the method of *sampling* [70]. The idea is to apply the algorithm only on a selected set of observations that are considered to be representative for the whole data trace. Usually, a random process is employed to select the observations out of the whole trace (*random sampling*) because a *systematic sampling*, for example by selecting every n-th observation, would be too vulnerable to periodicities in the trace. Often, a Poisson process is used as random process because it yields independent selections (*Poisson sampling*).

When random sampling is applied on a heavy-tail distributed data set it quickly shows that the obtained results exhibit a large variance. This is caused by the nature of the heavy-tailedness: such a data set comprises many small and only few, but very large values. Using random sampling, there is a high probability to accidentally "overlook" some of the values located in the tail of the distribution. *Stratified sampling* aims to reduce the variance of the result [70]. To achieve this, the population is divided into strata and each stratum is sampled separately such that even rare values have a high probability to be selected.

We use the following algorithm to build the strata. It operates on the sorted list of observations. Starting with the smallest value, the observations are added to the first stratum until the squared coefficient of variation of the data in the stratum reaches a threshold c^2_{max}. Then, a new, empty stratum is created and the algorithm is applied to the remaining observations. As a consequence, the regions of the distribution with low variability are assigned to large strata, whereas the regions with high variability are divided into many small strata. By selecting a fixed number of observations from each stratum, this approach makes that the regions with high variability are over-represented in the sampling result.

We have applied the stratification algorithm on several heavy-tailed data traces (they are discussed in detail in Section 4.6). Figure 4.3 shows the size (number of elements) of the generated strata for the RWTH trace with $c^2_{max} = 0.001$. It shows that, even for such a low threshold c^2_{max}, only a small number of strata is generated. This is true for all three traces discussed in Section 4.6 and is caused by the fact that, although the distribution is heavy-tailed, most of the data is located in the head of the distribution.

Note that an important simplification of the algorithm follows from this obser-

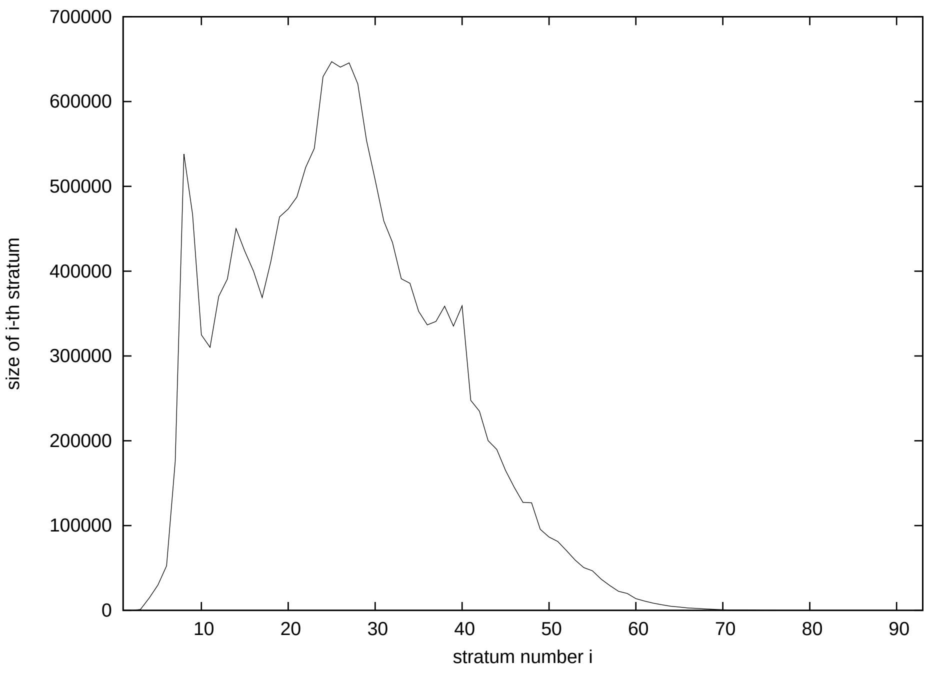


Figure 4.3: Size of the strata for a heavy-tailed distribution

vation. The threshold $c^2_{max} = 0.001$ means that the data contained in each stratum obeys a more or less deterministic distribution. For such strata, the random sampling actually is not required to select representative observations. Instead, it is sufficient to represent a stratum simply by the average of all values contained in the stratum. In this way, we obtain a small sequence of average values $\widehat{x}_1, \ldots, \widehat{x}_{N^*}$ which can be used as input data for the EM-algorithm. Since the strata have different sizes, the EM-algorithm has to weight each average $\widehat{x}_i$ by the weight $w_i = s_i/N$, where s_i is the size of stratum i and N is the total number of observations [90]. For HEDs, the EM-algorithm takes the form shown in Figure 4.4.

This means that the EM-algorithm with stratification (denoted as *sEM* in the following) becomes identical to the aggregation-based EM-algorithm [90] (see Section 4.4.2) for low threshold c^2_{max}. However, in the sEM-algorithm the number of strata and their size are determined dynamically.

4.6 Application and validation

In order to evaluate our fitting approaches, we applied them to three data traces. Using these, we made comparisons between:

1. first-order statistics of the obtained HEDs and those of the original data traces;
2. performance results for an M|G|1|K queue where the original data traces and the fitted distributions are used as service time distribution.

1. Build the strata with threshold c^2_{max}.
2. Compute for each stratum i, $i = 1, \dots, N^*$, the average $\widehat{x}_i$ and weight w_i.
3. Select an appropriate number of distributions I and initial parameters $\alpha = (c_1, \dots, c_I, \lambda_1, \dots, \lambda_I)$, as well as a positive required accuracy ε.
4. Compute for $i := 1$ to I:
   (a) $\delta(i|\widehat{x}_n, \alpha) = \frac{c_i \cdot p(\widehat{x}_n|\lambda_i)}{p(\widehat{x}_n|\alpha)}$,
   (b) $c'_i = \sum_{n=1}^{N^*} w_n \delta(i|\widehat{x}_n, \alpha)$,
   (c) $\lambda'_i = \frac{\sum_{n=1}^{N^*} w_n \delta(i|\widehat{x}_n, \alpha)}{\sum_{n=1}^{N^*} w_n \delta(i|\widehat{x}_n, \alpha) \cdot \widehat{x}_n}$.
5. Return to step 4 with $c_i := c'_i$ and $\lambda_i := \lambda'_i$ until the difference between c_i and c'_i and/or the difference between λ_i and λ'_i, for all i, is smaller than the accuracy boundary ε.

Figure 4.4: EM-algorithm with stratification for low threshold c^2_{max}

trace	number of entries	min	max	mean	median	SCV
RWTH	$17.3 \cdot 10^6$	118	10^7	6663.69	2638	6.12
NASA	$3.1 \cdot 10^6$	3	$6.8 \cdot 10^6$	20744.90	4142	13.39
Weibull	10^6	10^{-4}	$2.9 \cdot 10^7$	10000	1185	10

Table 4.2: Statistics for the data traces

The data traces are described in detail in Section 4.6.1. The obtained HEDs are discussed in Section 4.6.2. In Section 4.6.3, we present the results for the M|G|1|K queue.

4.6.1 Statistics of the data traces

Some important statistics of the studied data traces are summarized in Table 4.2. The squared coefficient of variation is denoted as SCV. We discuss each trace in the following.

RWTH trace

Early 2000, we collected the access logs of the RWTH Aachen proxy server. The logs comprise the description of about 115 million `HTTP` and `FTP` requests made over a period of 54 days. After some preprocessing and filtering, which comprises, e.g., the removal of incompletely processed requests, about 17.3 million requests of interest remained. We studied the sizes of the objects requested by the clients. Figure 4.5

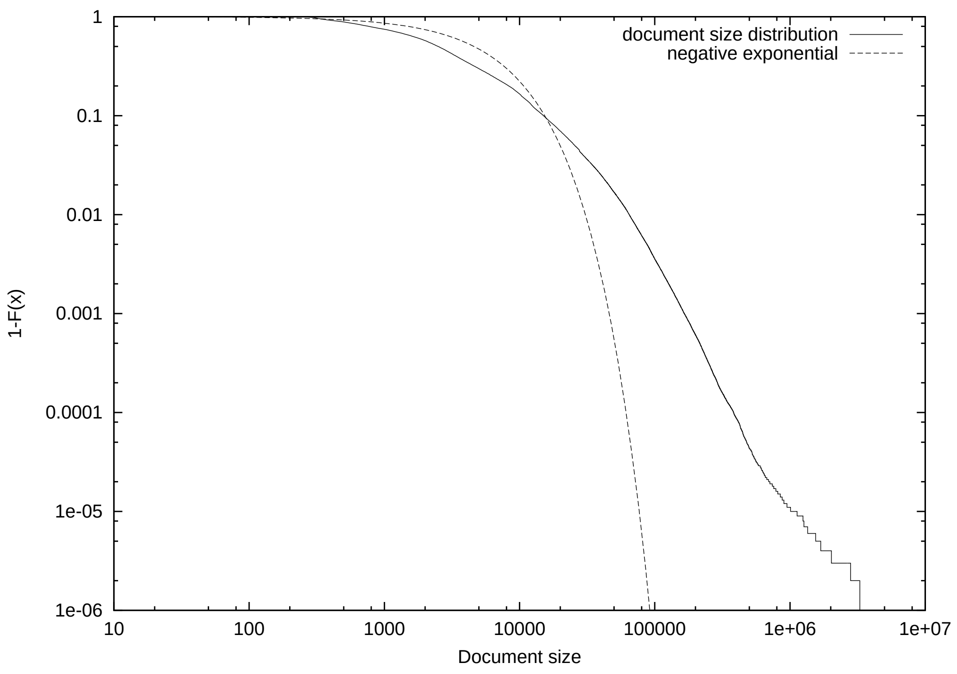


Figure 4.5: Complementary log-log plot of document size distribution (RWTH trace)

shows the complementary distribution of the object sizes as log-log plot. Obviously, the object-size distribution function decays much slower than a negative exponential distribution and is clearly heavy tailed. This is confirmed by the observation that the median of the distribution is much smaller than the mean.

NASA trace

The NASA trace was first presented and evaluated in 1996 by Arlitt and Williamson [5]. It consists of about 3.1 million requests collected at the web server of the Kennedy Space Center. As in the RWTH trace, the size distribution of the requested objects in the NASA trace is clearly heavy tailed, yielding a high SCV and a mean much larger than the median. Figure 4.6 shows the complementary distribution of the object sizes as log-log plot. Again, we observe that the object-size distribution function decays much slower than a negative exponential distribution.

Weibull trace

The Weibull trace has been artificially generated using a Weibull distribution with mean 10000 and SCV 10. Figure 4.7 shows the complementary distribution of the Weibull distribution as log-log plot.

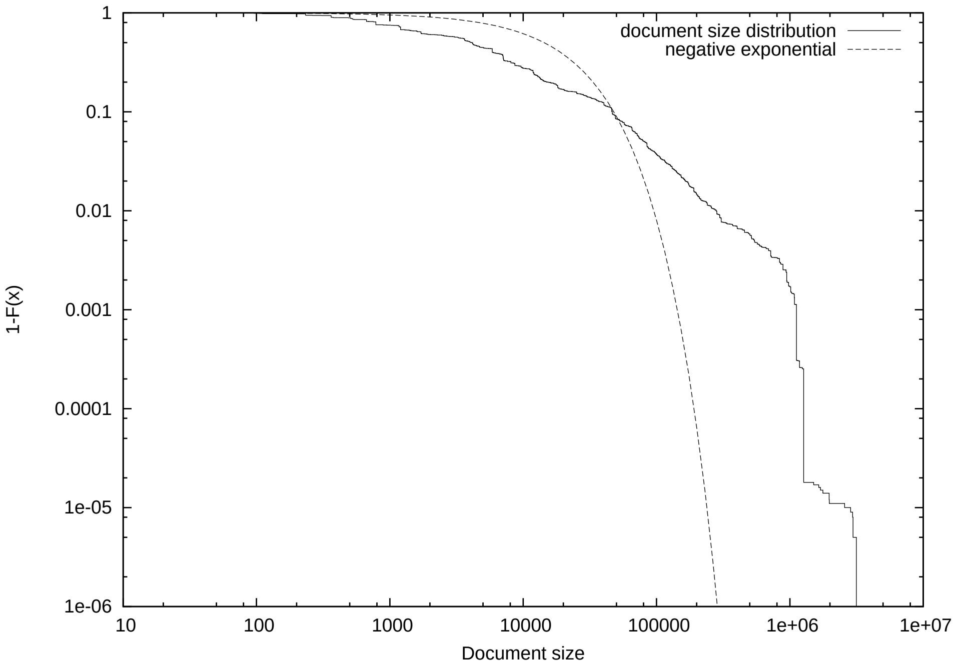


Figure 4.6: Complementary log-log plot of document size distribution (NASA trace)

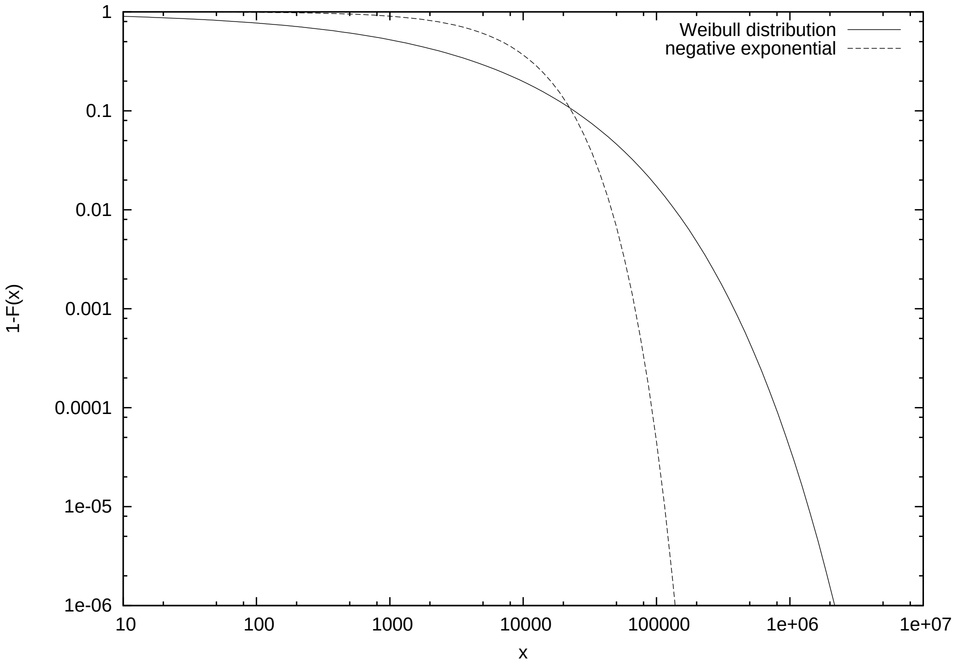


Figure 4.7: Complementary log-log plot of the Weibull distribution

measure	trace	Weibull	Pareto	EM:H_5	EM:H_{10}	sEM:H_5	sEM:H_{10}
E[X]	6663.69	6663.69	6663.69	6663.69	6663.69	6663.69	6663.69
SCV	6.12	6.12	6.12	6.23	6.23	6.22	6.22
rel. error	–	0%	0%	1.8%	1.8%	1.6%	1.6%
skewness	60.10	7.75	n.d.	62.20	63.33	62.17	62.09
rel. error	–	-87.1%	n.d.	3.5%	5.4%	3.4%	3.3%
median	2638	1322	4826	2641	2641	2643	2643
iterations	–	–	–	33	33	33	33

Table 4.3: Comparison of the statistics of the measurement data and fitted distributions (RWTH trace)

4.6.2 Matching HTDs

We have fitted different distributions to the data traces. For the empirical traces, a Weibull and Pareto distribution has been fitted by matching the first and second moment of the distribution. HEDs with 5 and 10 phases have been fitted using the EM-algorithm (from Section 4.3) and the sEM-algorithm (from Section 4.5). For the Weibull trace, HEDs with 10 and 20 phases have been fitted. The resulting HEDs are denoted as EM:H_5, EM:H_{10}, EM:H_{20} for the EM-algorithm, respectively, sEM:H_5, sEM:H_{10}, and sEM:H_{20} for the sEM-algorithm. The statistics of the fitted distributions are discussed in the following.

RWTH trace

The results for the RWTH trace are shown in Table 4.3. We have also included the statistics of the trace, the relative errors of the SCV and the skewness (always relative to the respective value of the trace), and the number of iterations required by EM and sEM to reach a fixed precision. We observe that the Weibull and the Pareto distribution fail to match the skewness (where available) and the mean of the data trace. In contrast, the HEDs match all statistics of the trace very well. No significant difference can be observed between the HEDs with 5 phases and those with 10 phases.

Table 4.3 furthermore shows that EM and sEM yield nearly identical results with the same number of iterations. The fact that sEM has to process less data per iteration (93 entries instead of more than $17 \cdot 10^6$ in the original trace) results in a speed-up of approximately 186000. However, this does not include the time to sort the data, as required by sEM.

measure	trace	Weibull	Pareto	EM:H_5	EM:H_{10}	sEM:H_5	sEM:H_{10}
E[X]	20744.9	20744.9	20744.9	20744.9	20744.9	20744.9	20744.9
SCV	13.39	13.39	13.39	14.43	14.43	14.42	14.42
rel. error	–	0%	0%	7.8%	7.8%	7.7%	7.7%
skewness	10.64	14.60	n.d.	12.73	12.73	12.73	12.73
rel. error	–	37.2%	n.d.	19.6%	19.6%	19.6%	19.6%
median	4142	1762	15008	3718	3718	3720	3720
iterations	–	–	–	78	78	78	78

Table 4.4: Comparison of the statistics of the measurement data and fitted distributions (NASA trace)

NASA trace

The results for the NASA trace are shown in Table 4.4. The Weibull and the Pareto distribution fail to match the skewness and the median. The results for the HEDs are better, but not as good as for the RWTH trace. This illustrates the fact that EM-based algorithms try to find an optimal *global* solution instead of focusing on specific moments of the distribution. Again, no significant differences can be observed between the fitted HEDs. The sEM-algorithm has to process 83 entries per iteration ($3.1 \cdot 10^6$ in the original trace) which results in a speed-up of approximately 3700.

Weibull trace

The results for the Weibull trace are shown in Table 4.5. Note that EM and sEM were not able to reach the desired precision with HEDs with 5 phases. The fitted HEDs match very well the statistics of the trace. Only for the skewness, we observe better results for the HEDs fitted by the sEM-algorithm. The sEM-algorithm has to process 1259 elements per iteration. This, in comparison with the other two traces, large number is caused by the artificial nature of the trace. Unlike a true measurement trace, it contains many unique values.

4.6.3 Embedding HTDs in queueing models

We have used the fitted distributions as service-time distribution in an M|G|1|100 queue. We have studied the mean length E[N] of the queue for two different offered loads (0.7 and 0.9). For the measurement data and for the Weibull distribution, the results were computed using a trace-driven and a stochastic discrete-event simulation. For the fitted HEDs, the results were obtained by numerical analysis of the M|HED|1|100 queue using the FiFiQueues tool (see Chapter 5 and 8). The Pareto distribution has not been evaluated here due to the obvious mismatch to the data traces, as discussed in the previous section.

measure	trace	EM:H_{10}	EM:H_{20}	sEM:H_{10}	sEM:H_{20}
E[X]	10000	10000	10000	10000	10000
SCV	10	9.99	9.99	10	10
rel. error	–	-0.1%	-0.1%	0%	0%
skewness	11.36	10.19	10.19	11.08	11.08
rel. error	–	-10.3%	-10.3%	-2.5%	-2.5%
median	1185	1164	1164	1141	1141
iterations	–	107	107	109	109

Table 4.5: Comparison of the statistics of the measurement data and fitted distributions (Weibull trace)

	measure	trace	Weibull	EM:H_5	EM:H_{10}	sEM:H_5	sEM:H_{10}
load=0.7	E[N]	5.60	6.56	5.69	5.69	5.68	5.68
	rel. error	–	17.1%	1.6%	1.6%	1.4%	1.4%
load=0.9	E[N]	21.00	24.65	21.08	21.08	21.07	21.07
	rel. error	–	17.4%	0.4%	0.4%	0.3%	0.3%

Table 4.6: Mean queue length for different offered loads (RWTH trace)

RWTH trace

The results for the RWTH trace are shown in Table 4.6. All HEDs provide accurate (and nearly identical) results. The Weibull distribution yields quite large errors which is caused by its considerably different skewness.

NASA trace

Table 4.7 shows the results for the NASA trace. All distributions provide good results. Since the HEDs do not match the SCV of the NASA trace very well, larger (but still small) errors can be observed for them in comparison to the RWTH trace.

	measure	trace	Weibull	EM:H_5	EM:H_{10}	sEM:H_5	sEM:H_{10}
load=0.7	E[N]	12.02	11.44	12.48	12.48	12.48	12.48
	rel. error	–	-4.8%	3.8%	3.8%	3.8%	3.8%
load=0.9	E[N]	32.57	31.16	32.23	32.23	32.23	32.23
	rel. error	–	-4.3%	-1.0%	-1.0%	-1.0%	-1.0%

Table 4.7: Mean queue length for different offered loads (NASA trace)

	measure	trace	EM:H_{10}	EM:H_{20}	sEM:H_{10}	sEM:H_{20}
load=0.7	E[N]	9.40	9.52	9.52	9.49	9.49
	rel. error	–	1.3%	1.3%	1.0%	1.0%
load=0.9	E[N]	29.27	29.64	29.64	29.34	29.34
	rel. error	–	1.3%	1.3%	0.2%	0.2%

Table 4.8: Mean queue length for different offered loads (Weibull trace)

Weibull trace

The results for the Weibull trace are shown in Table 4.8. All HEDs provide accurate (and nearly identical) results.

4.7 Summary and conclusions

In this chapter we have presented a direct way of fitting HEDs to heavy-tail distributed measurement data. The approach is based on the EM-algorithm and does not require any intermediate distribution function (form), and hence, delivers good approximations to the measurement data in cases where no closed-form HTD function is available. The thus-obtained HEDs match the first and second moment as well as higher moments and shape characteristics like the median of the original distribution very well. We have shown that this improves the quality of queueing analysis results.

In order to increase the efficiency of the method, we have also presented an extension of the EM-algorithm that applies sampling and stratification techniques to the data. We have observed that the new method yields identical results in the experiments, while exhibiting a speed-up of several magnitudes. We have shown that the extension is, in fact, a generalization of the aggregation-based EM-algorithm presented in [90] when applied to heavy-tail distributed measurement data; but it has the advantage to dynamically adapt some of its parameters that were assumed fixed in [90].

Chapter 5

FiFiQueues

In this chapters we discuss decomposition-based analysis methods that are based on so-called *first-order* traffic descriptors. Such traffic descriptors only contain information (first-order statistics) about the inter-arrival time *distribution* of the underlying traffic stream. First-order traffic descriptors are of special interest for two reasons. First, they are very intuitive: first-order statistics like mean arrival rate can be easily interpreted and visualized. Secondly, they are mathematically well understood and various algorithms are available for the computation of important performance measures like station load, mean queue length, etc., given such descriptors.

The focus of this chapter is on our network analyzer FiFiQueues, but we begin with two well-known methods that can be regarded as "ancestors" of FiFiQueues. In Section 5.1, we briefly describe Jackson queueing networks, followed by Whitt's Queueing Network Analyzer in Section 5.2. Like FiFiQueues, these two methods are based on first-order traffic descriptors. We show that they can be elegantly described in the context of our framework. In Section 5.3, we explain the algorithms employed in FiFiQueues. We present results concerning the existence of a fixed point for the fixed-point iteration of FiFiQueues in Section 5.4. The traffic descriptors of FiFiQueues rely on the first and second moment of the inter-arrival time of the traffic streams. Possible extensions to higher moments are discussed in Sections 5.5 and 5.6. Finally, this chapter concludes with a summary in Section 5.7.

5.1 Jackson queueing networks

The simplest open queueing networks allowing feedback are the so-called Jackson queueing networks (JQNs). Their analytical performance evaluation was developed by J.R. Jackson [54] in the 1950s.

5.1.1 The model class

In JQNs, all nodes are assumed to be infinite-buffer M|M|1 queues with the First-Come-First-Served (FCFS) service discipline. In many modeling applications, the restriction to Poisson arrival and service processes cannot be justified. However, the JQN model class already shows all features described in the framework (see Section 2.4), i.e., open queueing networks fed by external arrival processes, separate analysis of independent queueing stations, and routing with Markovian splitting. In fact, JQNs can be seen as the simplest (with regard to its analysis) but still reasonable class of open queueing networks.

5.1.2 The traffic descriptor

In JQNs, all traffic processes (including the external arrival processes) are assumed to be Poisson, hence a sufficient traffic descriptor only contains the arrival rate λ of the traffic stream, denoted as $\langle\lambda\rangle$.

5.1.3 Superposition of traffic streams

Merging two (possibly dependent) traffic streams does not necessarily yield a new Poisson stream. However, it can be shown that the nodes of a JQN still can be described by M|M|1 queues even when traffic merging occurs. Thus, to merge n traffic streams specified by $\langle\lambda_1\rangle, \ldots, \langle\lambda_n\rangle$ into one traffic stream $\langle\lambda\rangle$, one simply adds the rates:

$$\lambda = \sum_{i=1}^{n} \lambda_i.$$

5.1.4 Splitting traffic streams

The Markovian splitting of a Poisson stream $\langle\lambda\rangle$ again results into n Poisson streams. Let $p_1, \ldots, p_n$ be the splitting probabilities, then the resulting streams $\langle\lambda_1\rangle, \ldots, \langle\lambda_n\rangle$ are given by

$$\lambda_i = p_i \cdot \lambda, \qquad i = 1, \ldots, n.$$

5.1.5 The service operation

Let $\langle\lambda_A\rangle$ be the arrival traffic descriptor of the node, and μ its service rate. We require that $\lambda_A < \mu$, otherwise the station is not stable. Burke [18] proved that the departure process for a stable single server M|M|1 queue is a Poisson process with rate λ_A, and the departure process therefore can be described as $\langle\lambda_D\rangle$ with $\lambda_D = \lambda_A$.

5.1.6 Node performance

In this and the next section, we only give some examples of performance measures that can be computed for JQNs.

Let $\langle \lambda_A \rangle$ be the arrival traffic descriptor, and μ the service rate of the node. Then $\rho = \lambda_A/\mu$ is the utilization of the node. Since the node is an M|M|1 queue, the steady-state probability p_j to find j customers in the queue can be easily derived from the underlying birth-death Markov chain [46]:

$$p_j = (1-\rho)\rho^j, \qquad j = 0, 1, \ldots$$

Having computed the steady-state probabilities, quantities like the expected number of jobs in the queueing station E[N] can be calculated as

$$\mathrm{E}[N] = \sum_{j=0}^{\infty} j \cdot p_j = \frac{\rho}{1-\rho}.$$

Then, Little's law can be applied to compute the expected waiting time E[W].

Similarly, higher moments of measures can be computed too, e.g., the variance of the number of customers in the node:

$$\mathrm{Var}[N] = \sum_{j=0}^{\infty} (j - \mathrm{E}[N])^2 \cdot p_j = \frac{\rho}{(1-\rho)^2}.$$

5.1.7 Network performance

Since no losses occur and all nodes are required to be stable, the total throughput λ_{thr} of the network, i.e., the average number of customers passing through the network per time unit, is simply the sum of arrival rates $\lambda_{ext,i}$ of the external arrival processes:

$$\lambda_{thr} = \sum_{i=1}^{n} \lambda_{ext,i}$$

where $\langle \lambda_{ext,i} \rangle$ is the external traffic arriving at node i and n is the number of nodes. Other performance measures may be derived from the node performance measures. If $\lambda_{A,i}$ is the total amount of traffic arriving at node i, the expected number of visits E[V_i] of a customer at node i is given by [115, Eq. (77)]:

$$\mathrm{E}[V_i] = \lambda_{A,i}/\lambda_{thr}.$$

The expected total sojourn time E[T_{total}], i.e., the time a customer spends in the network, defined as the sum of the expected sojourn times E[T_i] at each node i, thus equals

$$\mathrm{E}[T_{total}] = \sum_{i=1}^{n} \mathrm{E}[T_i] = \sum_{i=1}^{n} \mathrm{E}[V_i] \left(\frac{1}{\mu_i} + \mathrm{E}[W_i] \right).$$

Since the total number of customers N_{total} in the network is the sum of customers present in each queueing station, we have

$$\mathrm{E}[N_{total}] = \sum_{i=1}^{n} \mathrm{E}[N_i],$$

where $\mathrm{E}[N_i]$ is the expected number of jobs in node i.

5.1.8 Complexity

Traffic computation

One can easily see that no fixed-point iteration scheme is needed to compute the traffic streams in a JQN. If $\mathbf{\Gamma} = (r_{ij})$ is the routing matrix, the traffic $\langle \lambda_{A,i} \rangle$ arriving at node i is given by the so-called *first-order traffic equation*:

$$\lambda_{A,i} = \lambda_{ext,i} + \sum_{j=1}^{n} \lambda_{D,j} \cdot r_{ji}.$$

Since $\lambda_{D,i} = \lambda_{A,i}$, the traffic equations form a system of linear equations which can be expressed in vector/matrix notation as $\boldsymbol{\lambda}_{\mathbf{A}} = \boldsymbol{\lambda}_{\mathbf{ext}} + \boldsymbol{\lambda}_{\mathbf{A}} \cdot \mathbf{\Gamma}$, or, after transformation, as

$$\boldsymbol{\lambda}_{\mathbf{A}} = \boldsymbol{\lambda}_{\mathbf{ext}} (\mathbf{I} - \mathbf{\Gamma})^{-1}.$$

Thus, to find $\boldsymbol{\lambda}_{\mathbf{A}}$ we solve the linear system

$$\boldsymbol{\lambda}_{\mathbf{A}} (\mathbf{I} - \mathbf{\Gamma}) = \boldsymbol{\lambda}_{\mathbf{ext}}.$$

This system of equations can be solved by direct methods like Gaussian elimination, resulting in a time complexity of $O(n^3)$, or by iterative methods like Gauss-Seidel. For very large networks, we can make use of the fact that the routing matrix typically is a sparse matrix. In this way, the time complexity of an iterative solver such as Gauss-Seidel can be reduced to about $O(c \cdot n)$ where c is the average number of outgoing connections per station.

Node performance and network performance computation

The expressions given in Section 5.1.6 for the node performance measures can be computed in constant time for each node. For the network performance, most results require summation over the number of nodes in the network which yields a time complexity of $O(n)$.

5.2 Whitt's Queueing Network Analyzer

In the early 1980s, Whitt presented the Queueing Network Analyzer (QNA) [115, 116], a software package developed at Bell Laboratories for the approximate analysis of open queueing networks. Unlike prior approaches which were based on Markovian models, QNA allows for the analysis of open queueing networks where the external arrival processes need not be Poissonian and the service times need not be negative exponentially distributed. Additionally, QNA is able to perform the analysis fast: due to the involved approximations and assumptions, the network traffic analysis is, in essence, reduced to the solution of a set of linear equations, comparable to those in JQNs.

In the following, we will give an overview of the functionality of QNA. The structure of our presentation will slightly differ from Whitt's original paper [115] since we embed the QNA approach in the framework described in Section 2.4.

5.2.1 The model class

Since our framework originally has been based on QNA's design, its model class naturally fits the framework, i.e.,

- open queueing networks fed by external arrival processes,
- separate analysis of independent queueing stations, and
- a routing matrix with Markovian splitting.

The nodes are GI|G|m multiserver queues without capacity constraints and with the FCFS service discipline. The external arrival processes as well as the service processes of the nodes are described by the first and the second moment of the inter-arrival, resp. service time distributions.

In addition to the framework, QNA's model class includes three features which we will not describe in the following. First, QNA is able to analyze networks with multiple classes of customers, and secondly, networks with immediate feedback are allowed, i.e., with routing probabilities $r_{ii} \neq 0$. Both features are "implemented" by adding a pre-processing and post-processing phase to the core QNA algorithms. The third feature, the customer multiplication factor of a node, only requires small modifications in the service operation equations.

5.2.2 The traffic descriptor

As already explained, the external arrival processes are specified by the first and second moment of the inter-arrival times. In fact, this representation is also applied to the traffic streams between the nodes. More specifically, QNA uses the traffic descriptor $\langle \lambda, c^2 \rangle$ to describe a traffic stream where λ is the arrival rate and c^2 is the squared coefficient of variation of the inter-arrival time.

Clearly, this allows the representation of non-Poissonian processes. However, neither higher moments nor correlations of the arrival stream are considered, which may influence the quality of the analysis. QNA employs fine-tuned heuristics deduced from simulation studies to reduce the errors introduced by this simplification.

5.2.3 Superposition of traffic streams

To merge n traffic streams specified by $\langle \lambda_1, c_1^2 \rangle, \ldots, \langle \lambda_n, c_n^2 \rangle$ into one traffic stream $\langle \lambda, c^2 \rangle$, QNA first computes the total arrival rate which is simply given by

$$\lambda = \sum_{i=1}^{n} \lambda_i.$$

As already mentioned, QNA's efficiency is based on the fact that it computes the traffic descriptors by linear equations. The above expression for λ is clearly linear in λ_i. For c^2 a linear equation can be found, too, by the <u>as</u>ymptotic approximation method (AS):

$$c_{AS}^2 = \sum_{i=1}^{n} \frac{\lambda_i}{\lambda} c_i^2.$$

However, the asymptotic method does not work well for a wide range of cases. It therefore is combined with the <u>s</u>tationary-<u>i</u>nterval method (SI) resulting into the hybrid approximation

$$c^2 = w \cdot c_{AS}^2 + (1 - w) \cdot c_{SI}^2.$$

The stationary-interval method does not provide a linear expression for c_{SI}^2, but experiments have shown that setting c_{SI}^2 to 1 increases the average error only by 1 percent, hence we obtain

$$c^2 = w \cdot c_{AS}^2 + (1 - w).$$

Simulations have shown that the above approximations do impact the quality of the analysis of a node which takes the merged traffic stream as input. To improve the results, QNA respects the utilization ρ of the node in the computation of the factor w. With $\rho = \lambda / \mu$ (where μ is the service rate of the queueing station), QNA sets

$$\begin{aligned} w &= \left[1 + 4(1-\rho)^2(v-1)\right]^{-1} \\ \text{with } v &= \left(\sum_{i=1}^{n} \left(\frac{\lambda_i}{\lambda} \right)^2 \right)^{-1}. \end{aligned}$$

5.2.4 Splitting traffic streams

When splitting, QNA assumes that the involved processes are renewal processes. Under this assumption, an exact solution is available. For n splitting probabilities $p_1, \ldots, p_n$ and the traffic stream $\langle \lambda, c^2 \rangle$, we obtain the splitted streams

$$\langle \lambda_1, c_1^2 \rangle, \ldots, \langle \lambda_n, c_n^2 \rangle,$$

with

$$\lambda_i = p_i \cdot \lambda, \quad \text{and} \quad c_i^2 = p_i \cdot c^2 + (1 - p_i), \qquad i = 1, \ldots, n.$$

5.2.5 The service operation

As explained in Section 5.2.1, network nodes are analyzed as GI|G|m queues. Let $\langle \lambda_A, c_A^2 \rangle$ be the arrival traffic descriptor of the node and m the number of service entities. The service process is specified by the service rate μ and by the squared coefficient of variation c_S^2 of the service time distribution. We require the stability of all stations, i.e., $\lambda_A < \mu$. How does QNA compute the departure descriptor $\langle \lambda_D, c_D^2 \rangle$?

Since the queues are stable and have infinite capacity, no losses occur and we clearly have $\lambda_D = \lambda_A$. To compute c_D^2, Whitt combines Marshall's formula [79] with other approximations to obtain

$$c_D^2 = 1 + (1 - \rho^2)(c_A^2 - 1) + \frac{\rho^2}{\sqrt{m}}(c_S^2 - 1). \tag{5.1}$$

The involved approximations may lead to large errors when c_S^2 is small, thus QNA uses the following extension of the above formula:

$$c_D^2 = 1 + (1 - \rho^2)(c_A^2 - 1) + \frac{\rho^2}{\sqrt{m}}(\max\{c_S^2, 0.2\} - 1). \tag{5.2}$$

Note again the linearity of the expressions for λ_D and c_D^2 in the arrival traffic $\langle \lambda_A, c_A^2 \rangle$.

5.2.6 Node performance

QNA is able to compute results for the first and second moment of the waiting time W and the queue length N. Due to the complexity of the involved approximations, we limit our presentation only to the simplest one, i.e., the computation of E[W] in the case of single-server GI|G|1 queues. The required computation steps for the other quantities can be found in [115, Eq. (46)–(71)]. For given arrival traffic $\langle \lambda_A, c_A^2 \rangle$, service descriptor $\langle \mu, c_S^2 \rangle$ and utilization ρ, E[W] is approximated as

$$\mathrm{E}[W] = \frac{\rho}{2(1 - \rho)\mu}(c_A^2 + c_S^2) g(\rho, c_A^2, c_S^2), \tag{5.3}$$

where the function ρ is defined as

$$g(\rho, c_A^2, c_S^2) = \begin{cases} \exp\left(\frac{2(1-\rho)(1-c_A^2)^2}{3\rho(c_A^2+c_S^2)}\right), & c_A^2 < 1, \\ 1, & c_A^2 \geq 1. \end{cases}$$

Note that Equation (5.3) is exact for $c_A^2 = 1$, i.e., in the case of a M|G|1 queue. When $c_A^2 < 1$, it is equivalent to the Krämer and Langenbach-Belz approximation [63].

5.2.7 Network performance

The results presented for network performance measures in Jackson queueing networks (see Section 5.1.7) can also be applied to QNA, providing expressions for $\mathrm{E}[V_i]$, $\mathrm{E}[T_i]$, $\mathrm{E}[T_{total}]$ and $\mathrm{E}[N_{total}]$. Additionally, Whitt developed approximations for the variances of the above stated measures [115, Eq. (80)–(84)].

5.2.8 Complexity

In the above sections, we have repeatedly pointed out the linearity of the employed equations for the three traffic operations merging, splitting, and service. In fact, QNA uses this linearity to efficiently evaluate the queueing network.

First, for the arrival rates of the traffic streams the system of equations derived for JQNs (Section 5.1.8) is also valid for QNA. Let $\langle \lambda_{A,i}, c_{A,i}^2 \rangle$ be the traffic arriving at node i, $\langle \lambda_{D,i}, c_{D,i}^2 \rangle$ the traffic leaving this node and $\langle \lambda_{ext,i}, c_{ext,i}^2 \rangle$ the external traffic. If $\mathbf{\Gamma} = (r_{ij})$ is the routing matrix, the following traffic equation holds for each node $i = 1, \ldots, n$ of the network:

$$\lambda_{A,i} = \lambda_{ext,i} + \sum_{j=1}^{n} \lambda_{D,j} \cdot r_{ji} \tag{5.4}$$

Again, QNA's model class implies $\lambda_{D,i} = \lambda_{A,i}$ and the traffic equations form a system of linear equations which can be expressed in vector/matrix notation (as for JQNs):

$$\boldsymbol{\lambda}_{\mathbf{A}} = \boldsymbol{\lambda}_{\mathbf{ext}}(\mathbf{I} - \mathbf{\Gamma})^{-1}.$$

For the squared coefficients of variation of the traffic streams a system of equations can be set up, too. The synthesis of the superposition and the splitting operations yields

$$c_{A,i}^2 = (1 - w_i) + w_i \left(p_{ext,j} c_{ext,i}^2 + \sum_{j=1}^{n} p_{j,i}(r_{ji} c_{D,j}^2 + 1 - r_{ji}) \right)$$

where $p_{j,i} = \lambda_{D,j} r_{j,i} / \lambda_{A,i}$ is the fraction of traffic arriving from node j to node i and $p_{ext,j} = \lambda_{ext,i}/\lambda_{A,i}$ is the fraction of external traffic arriving to node i. Finally, if we include the result of the service operation we obtain the following system of linear equations

$$\begin{aligned} c_{A,i}^2 = (1-w_i) + w_i\{ \ & p_{ext,j} c_{ext,i}^2 \\ & + \sum_{j=1}^{n} p_{j,i} (r_{ji}(1 + (1-\rho_i^2)(c_{A,j}^2 - 1) \\ & + \frac{\rho_i^2}{\sqrt{m_i}} (\max(c_{S,i}^2, 0.2) - 1)) + 1 - r_{ji})\}. \end{aligned} \tag{5.5}$$

Using the two systems of linear equations formed by Equation (5.4) and Equation (5.5), the traffic descriptors can easily be computed. Thus, obviously QNA has the same time complexity as the Jackson network method (Section 5.1.8), modulo some constant factors, but provides much more accurate results.

5.3 FiFiQueues

In the mid-1990's Haverkort and Weerstra [43, 44, 45, 113] extended Whitt's QNA approach by means of replacing the core of the analysis: the service operation. Unlike QNA, their new approach, called QNAUT, does not use the descriptor of the arrival traffic directly to compute the departure traffic descriptor, but assumes that the arrival traffic descriptor can be used to construct a phase-type (PH) renewal process (see Section 3.2) which approximates the "real" underlying arrival process. This allows for the inclusion of finite-buffer queueing stations as well as for the analysis of the queueing stations by matrix-geometric and general Markovian techniques, instead of the approximations used originally in QNA.

At the end of the 1990s, we developed an extended version of the original approach in that we removed a few approximate steps and enhanced the model class [100, 101, 103]. In particular, this enhanced class provides:

- exact results for the the departure process based on the results of Bocharov [13] for PH|PH|1|K queues;
- efficient per-queue analysis;
- for each finite queueing station, a traffic stream is computed which consists of the customers rejected at a completely filled queue. This loss traffic stream can be used as arrival stream for other queueing stations like any other "regular" departure traffic stream.

This approach, as well as the analysis tool developed from it, is named *FiFiQueues* (*Fi*xpoint-based analysis of networks with *Fi*nite *Queues*).

5.3.1 The model class

Clearly, the model class fulfills the specifications of the framework. The external arrival processes are described, as in QNA, by the first and the second moment of the inter-arrival time. The main differences to QNA's model class are:

- the service processes are specified by PH renewal processes;
- the queueing stations can have *infinite* or *finite* queueing capacities. The nodes are analyzed as PH|PH|1(|K) queues with the FCFS service discipline. The customer multiplication factor known from QNA is also supported, but not described in the following;
- finite queues have two output streams: the "regular" departure traffic stream and the loss traffic stream which consists of the customers rejected by a full queue.

Seen from a single queue, customers arriving at a completely filled queue are simply lost. This form of blocking is common in communication networks (communication blocking) and has an important advantage: unlike other types of blocking (like back-blocking), it still allows the independent analysis of the queueing stations (see Section 2.2.2).

Just like the regular departure traffic of a queueing station with finite capacity, the loss traffic is not known a priori and is computed by the analysis of the station. The "reuse" of loss traffic streams as arrival streams to other nodes requires an auxiliary routing matrix. Its handling will not be discussed further in the following sections, since, once the traffic descriptors of the loss streams are known, they can easily be processed like the regular departure traffic. However, note that loss traffic streams should only be used very carefully in feedback networks: if a loss traffic stream is fed back directly or indirectly to the node which produced the stream, it can prevent the iteration algorithm (see Section 2.3) to terminate because the arrival rate to the node increases in each iteration step.

5.3.2 The traffic descriptor

As in QNA (see Section 5.2.2) the external arrival processes as well as the inter-node traffic streams are described by the first and second moment of the inter-arrival times. The traffic descriptor $\langle \lambda, c^2 \rangle$ contains the arrival rate λ and the squared coefficient of variation c^2 of the inter-arrival time.

5.3.3 Superposition of traffic streams

To merge n traffic streams specified by $\langle \lambda_1, c_1^2 \rangle, \ldots, \langle \lambda_n, c_n^2 \rangle$ into one traffic stream $\langle \lambda, c^2 \rangle$, we adopt the hybrid approximation of QNA, i.e.,

$$\lambda = \sum_{i=1}^{n} \lambda_i, \tag{5.6}$$

$$c^2 = w \cdot \sum_{i=1}^{n} \frac{\lambda_i}{\lambda} c_i^2 + (1 - w), \tag{5.7}$$

with

$$w = \left[1 + 4(1-\rho)^2 (v - 1)\right]^{-1}, \quad \text{and} \quad v = \left(\sum_{i=1}^{n} \left(\frac{\lambda_i}{\lambda} \right)^2 \right)^{-1},$$

where ρ is the utilization of the node receiving the resulting traffic stream. It should be emphasized that these formulae where originally designed in the context of QNA's model class, i.e., *not* for finite queues. Thus, their usage in FiFiQueues introduces auxiliary errors to the computation, in addition to the errors inherent to the hybrid approximation method. In principle, it is possible to exactly compute the resulting arrival process in case all the streams to be merged are departure streams from finite queues: since finite queues are analyzed as PH|PH|1|K queues, an exact representation as MAP of each departure process is available (see Section 9.1), and consequently, also the resulting MAP exactly describing the superposition. *In practice*, the resulting MAP could be far too large (in number of states) for any further analysis. Therefore, FiFiQueues uses QNA's approximation equations instead.

One may wonder if we could obtain better results by not following QNA's linear approximation ($c_{SI}^2 = 1$) but in actually computing the correct value for c_{SI}^2. Our experiments have shown that nearly the same results are obtained by doing so. This is consistent with Whitt's observation that fixing c_{SI}^2 to 1 increases the average error only by 1 percent.

5.3.4 Splitting traffic streams

When splitting, we assume that the involved processes are renewal processes. Under this assumption, an exact solution is available. For n splitting probabilities $p_1, \ldots, p_n$ and the traffic stream $\langle \lambda, c^2 \rangle$, we obtain the splitted streams $\langle \lambda_1, c_1^2 \rangle, \ldots, \langle \lambda_n, c_n^2 \rangle$ by

$$\lambda_i = p_i \cdot \lambda, \quad \text{and} \quad c_i^2 = p_i \cdot c^2 + (1 - p_i), \qquad i = 1, \ldots, n. \tag{5.8}$$

5.3.5 The service operation

We already stated that the nodes are analyzed as PH|PH|1(|K) queues. Thus, before a queueing station can be analyzed we need to find a PH distribution that fits the two moments given in the arrival traffic descriptor. In the following we will explain the fitting step and the actual queueing analysis procedure, thereby treating PH|PH|1 and PH|PH|1|K queues separately. We require that the PH|PH|1 queues are stable, i.e., the total arrival rate at a PH|PH|1 station should be smaller than its service rate.

Phase-type representation of the arrival processes

Let $\langle \lambda, c^2 \rangle$ be the arrival traffic descriptor. We write $\mathrm{E}[X] = 1/\lambda$ for the corresponding mean inter-arrival time. Clearly, having only two moments allows us some freedom to select an appropriate PH distribution. We require that the chosen PH distribution, represented by $(\boldsymbol{\alpha}, \mathbf{A})$

1. matches the two moments exactly (at least for a certain range; see below), and
2. is as compact as possible, i.e., has the smallest number of transient states m.

Additionally, we want that the employed fitting procedure does not consume to much time since it has to be executed every time when a node is analyzed. In FiFiQueues, we use the following approach, first presented in [44]. Two cases are distinguished:

- In case $c^2 \leq 1$, we use a hypo-exponential distribution with $m = \lceil \frac{1}{c^2} \rceil$ phases and initial probability vector $\boldsymbol{\alpha} = (1, 0, \cdots, 0)$. The matrix $\mathbf{A}$ is then given as

$$\mathbf{A} = \begin{pmatrix} -\lambda_0 & \lambda_0 & & & \\ & -\lambda_1 & \lambda_1 & & \\ & & \ddots & \ddots & \\ & & & -\lambda_{m-2} & \lambda_{m-2} \\ & & & & -\lambda_{m-1} \end{pmatrix}, \tag{5.9}$$

  where $\lambda_i = m/\mathrm{E}[X]$, for $0 \leq i < m-2$ and where

$$\lambda_{m-1} = \frac{2m\left(1 + \sqrt{\frac{1}{2}m(mc^2 - 1)}\right)}{\mathrm{E}[X](m + 2 - m^2c^2)} \quad \text{and} \quad \lambda_{m-2} = \frac{m\lambda_{m-1}}{2\lambda_{m-1}\mathrm{E}[X] - m} \ .$$

  For small c^2, PH distributions with a large number of states will be obtained. To limit the computational requirements in the analysis process we do not allow c^2 to be smaller than $\frac{1}{10}$. This approximation corresponds to an Erlang-10 distribution and produces generally good results, also as approximation for deterministic distributions.

- In case $c^2 > 1$, we take a hyper-exponential distribution with $m = 2$ phases. Such a distribution has three free parameters: the choice probability p between the two possible phases and the rates μ_1 and μ_2 of the two phases. Fitting the first two moments thus leaves one degree of freedom. We resolve this by assuming so-called “balanced means”, meaning that the ratios p/μ_1 and $(1-p)/\mu_2$ should be equal. This then yields $\boldsymbol{\alpha} = (p, 1-p)$ and

$$\mathbf{A} = \begin{pmatrix} -\frac{2p}{\mathrm{E}[X]} & 0 \\ 0 & -\frac{2(1-p)}{\mathrm{E}[X]} \end{pmatrix} \quad \text{with} \quad p = \frac{1}{2} + \frac{1}{2}\sqrt{\frac{c^2-1}{c^2+1}} \ .$$

Analysis of PH|PH|1|K queues

The underlying CTMC. Let $(\boldsymbol{\alpha}, \mathbf{A})$ be the arrival PH renewal process with l states as obtained by the fitting step and $(\boldsymbol{\beta}, \mathbf{B})$ the service PH renewal process with m states. Then we can describe the behavior of a node with queueing capacity K by a QBD process [85] (see Section 3.4) with $K+1$ levels, where level 0 consists of l states and where levels 1 through K consist of $l \cdot m$ states each.

The i-th level represents the state of the system when it contains i customers. A step from level i to level $i+1$ ($i < K$) stands for an arrival and a step from level i to level $i-1$ ($i > 0$) stands for a departure. The $l \cdot m$ states of a level $i > 0$ describe the current state of the arrival and of the service processes (level 0 contains only l states because the queue is empty and the service process has not yet started; it only records the state of the arrival process). This leads to the following generator matrix of the Markov chain:

$$\mathbf{Q} = \begin{pmatrix} \mathbf{A} & \mathbf{A^0}\boldsymbol{\alpha} \otimes \boldsymbol{\beta} & & \\ \mathbf{I} \otimes \mathbf{B^0} & \mathbf{A} \oplus \mathbf{B} & \mathbf{A^0}\boldsymbol{\alpha} \otimes \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \otimes \mathbf{B^0}\boldsymbol{\beta} & \mathbf{A} \oplus \mathbf{B} & \mathbf{A^0}\boldsymbol{\alpha} \otimes \mathbf{I} \\ & \ddots & \ddots & \ddots \\ & & \mathbf{I} \otimes \mathbf{B^0}\boldsymbol{\beta} & (\mathbf{A} + \mathbf{A^0}\boldsymbol{\alpha}) \oplus \mathbf{B} \end{pmatrix},$$

where $\mathbf{A^0} = -\mathbf{A} \cdot \mathbf{1}$, $\mathbf{B^0} = -\mathbf{B} \cdot \mathbf{1}$, $\mathbf{L} \oplus \mathbf{M} = \mathbf{L} \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{M}$, and $\otimes$ is the Kronecker product operator (also known as tensor or matrix direct product operator).

The steady-state solution $\mathbf{v}$ of the Markov chain with generator $\mathbf{Q}$ can be obtained by solving the global balance equation (see Section 3.4):

$$\mathbf{v} \cdot \mathbf{Q} = \mathbf{0} \quad \text{and} \quad \mathbf{v} \cdot \mathbf{1} = 1.$$

The vector $\mathbf{v}$ is of size $l + K \cdot l \cdot m$. In the following we write $\mathbf{v_0}$ for the vector $(v_1, \ldots, v_l)$ which contains the steady-state probabilities of level 0 and we write $\mathbf{v_i}$ for the vector $(v_{l+1+(i-1)\cdot l\cdot m}, \ldots, v_{l+i\cdot l\cdot m})$ which contains the steady-state probabilities of level $i = 1, \ldots, K$.

The departure traffic. The steady-state solution vector $\mathbf{v}$ now allows us to compute the departure traffic descriptor $\langle \lambda_D, c_D^2 \rangle$. For this, we use the results of Bocharov presented in [13] which we will briefly describe in the following.

We begin with the computation of the blocking probability π, i.e., the probability that an arriving customer encounters a full queue and, hence, is lost. The vector $\mathbf{v_{A,K}}$ gives for this situation the state probabilities and it holds

$$\mathbf{v_{A,K}} = \frac{1}{\lambda_A}\mathbf{v_K}(\mathbf{A^0} \otimes \mathbf{I}),$$

where λ_A stands for the arrival rate to the node and K stands for the queueing capacity of the node. This leads to the blocking probability π:

$$\pi = \mathbf{v_{A,K}} \cdot \mathbf{1}.$$

With π, we easily find the departure rate of served customers as

$$\lambda_D = \lambda_A(1-\pi). \tag{5.10}$$

Higher moments of the inter-departure time can be computed using the following consideration. If the queue is not empty after a departure took place, the distribution of the time up to the next departure is equal to the distribution of the service time. Otherwise, it is equal to the distribution of the sum of the time until the next customer arrival and its service time (which are independent). The probability to leave an empty queue at departure instant $t+\varepsilon$ is

$$\mathbf{v_{D,0}} = \frac{1}{\lambda_D}\mathbf{v_1}(\mathbf{I} \otimes \mathbf{B^0}).$$

This leads Bocharov to the expression for the i-th moment d_i of the inter-departure time distribution:

$$d_i = b_i + \mathbf{v_{D,0}} \sum_{j=1}^{i} (-1)^j \frac{i!}{(i-j)!} \mathbf{A}^{-j}\mathbf{1} b_{i-j}, \tag{5.11}$$

where b_i is the i-th moment of the service time distribution. Thus, one can easily verify that the variance σ_D^2 of the departure process is

$$\sigma_D^2 = \sigma_S^2 + \sigma_0^2, \tag{5.12}$$

where σ_S^2 is the variance of the service time distribution and σ_0^2 equals

$$\sigma_0^2 = 2\mathbf{v_{D,0}}\mathbf{A}^{-2}\mathbf{1} - (\mathbf{v_{D,0}}\mathbf{A}^{-1}\mathbf{1})^2.$$

The squared coefficient of variation is then given by $c_D^2 = \lambda_D^2 \sigma_D^2$.

The loss traffic. The rate of loss λ_L is given by $\lambda_L = \lambda_A \cdot \pi$, where π is the loss probability. In oder to obtain higher moments of the inter-loss time we describe the loss process by the MAP $(\mathbf{L_0}, \mathbf{L_1})$ with

$$\mathbf{L_0} = \begin{pmatrix} \mathbf{A} & \mathbf{A^0}\boldsymbol{\alpha} \otimes \boldsymbol{\beta} & & \\ \mathbf{I} \otimes \mathbf{B^0} & \mathbf{A} \oplus \mathbf{B} & \mathbf{A^0}\boldsymbol{\alpha} \otimes \mathbf{I} & \\ \mathbf{0} & \mathbf{I} \otimes \mathbf{B^0}\boldsymbol{\beta} & \mathbf{A} \oplus \mathbf{B} & \mathbf{A^0}\boldsymbol{\alpha} \otimes \mathbf{I} \\ & \ddots & \ddots & \ddots \\ & & \mathbf{I} \otimes \mathbf{B^0}\boldsymbol{\beta} & \mathbf{A} \oplus \mathbf{B} \end{pmatrix}, \mathbf{L_1} = \begin{pmatrix} \mathbf{0} & & \\ & \ddots & \\ & & \mathbf{A^0}\boldsymbol{\alpha} \otimes \mathbf{I} \end{pmatrix}.$$

The underlying CTMC of this MAP is the CTMC of the QBD where arrivals in the last level K have been marked. Naturally, it has the same steady-state probability vector $\mathbf{v}$. The i-th moment of the inter-loss time is given by

$$\mathrm{E}[L^i] = \frac{i!}{\lambda_D} \mathbf{v}(-\mathbf{L_0})^{-(i-1)}\mathbf{1}, \tag{5.13}$$

hence, its second moment equals

$$\mathrm{E}[L^2] = \frac{2}{\lambda_L} \mathbf{v}(-\mathbf{L_0})^{-1}\mathbf{1}.$$

Analysis of PH|PH|1 queues

The underlying CTMC. Let $(\boldsymbol{\alpha}, \mathbf{A})$ be the arrival PH renewal process with l states and $(\boldsymbol{\beta}, \mathbf{B})$ the service PH renewal process with m states. Again, the behavior of the queue can be described by a QBD process with a generator matrix similar to the one of the PH|PH|1|K; the only difference is the fact that it has repeating columns ad infinitum:

$$\mathbf{Q} = \begin{pmatrix} \mathbf{A} & \mathbf{A^0}\boldsymbol{\alpha} \otimes \boldsymbol{\beta} & & \\ \mathbf{I} \otimes \mathbf{B}^0 & \mathbf{A} \oplus \mathbf{B} & \mathbf{A^0}\boldsymbol{\alpha} \otimes \mathbf{I} & \\ \mathbf{0} & \mathbf{I} \otimes \mathbf{B^0}\boldsymbol{\beta} & \mathbf{A} \oplus \mathbf{B} & \mathbf{A^0}\boldsymbol{\alpha} \otimes \mathbf{I} \\ & \ddots & \ddots & \ddots \end{pmatrix},$$

with the infinite steady-state probability vector $\mathbf{v}$ fulfilling

$$\mathbf{v} \cdot \mathbf{Q} = \mathbf{0} \quad \text{and} \quad \mathbf{v} \cdot \mathbf{1} = 1.$$

We refer to Section 3.3 for an overview of solution techniques.

The departure traffic. Since infinite queues produce no loss, we have

$$\lambda_D = \lambda_A, \tag{5.14}$$

where λ_A is the arrival rate to the node. The variance of the output stream is calculated using the same approach as in the case of finite-buffer queues, hence

$$\sigma_D^2 = \sigma_S^2 + \sigma_0^2, \tag{5.15}$$

with $\mathbf{v_{D,0}} = \frac{1}{\lambda_D}\mathbf{v_1}(\mathbf{I} \otimes \mathbf{B^0})$ and $\sigma_0^2 = 2\mathbf{v_{D,0}}\mathbf{A}^{-2}\mathbf{1} - (\mathbf{v_{D,0}}\mathbf{A}^{-1}\mathbf{1})^2$.

5.3.6 Node performance

FiFiQueues computes the first and second moment of the waiting time W and the queue length N. Again, queues with finite and infinite buffer capacity are treated separately.

Node performance of PH|PH|1|K queues

The j-moment $\mathrm{E}[N^j]$ of the queue length distribution (including the job in service) is given by

$$\mathrm{E}\left[N^j\right] = \sum_{i=1}^{K} i^j \mathbf{v_i} \mathbf{1}. \tag{5.16}$$

Hence, mean and variance of the queue length N are:

$$\mathrm{E}[N] = \sum_{i=1}^{K} i \cdot \mathbf{v_i}\mathbf{1} \quad \text{and } \mathrm{Var}[N] = \sum_{i=1}^{K} i^2 \cdot \mathbf{v_i}\mathbf{1} - \mathrm{E}[N]^2 .$$

Equation (4.4) in [13] gives the Laplace-Stieltjes transform of the waiting time probability density function. From this equation, any desired moment of the waiting time can be derived. For the mean and the variance we obtain [13, Eq. (4.5)–(4.7)]:

$$\begin{aligned} \mathrm{E}[W] &= \frac{1}{\lambda_D}(\mathrm{E}[N] - 1 + \mathbf{v_0}\mathbf{1}), \\ \mathrm{Var}[W] &= \frac{2}{\lambda_D}\left(\mu \cdot \mathbf{q_2}\mathbf{1} - \left(\mathbf{q_1}\left(\mathbf{1} \otimes \mathbf{B^{-1}}\mathbf{1}\right)\right)\right) - \mathrm{E}[W]^2 , \end{aligned}$$

where μ is the service rate. The components of the vector $\mathbf{q_1}$ resp. $\mathbf{q_2}$ give the first, resp. second binomial moment of the number of jobs in the queue as a function of the system state. For $j > 0$, the j-th binomial moment $\mathbf{q}_j$ is defined as [13, Eq. (3.1)]:

$$\mathbf{q}_j = \sum_{i=j+1}^{K} \binom{i-1}{j} \mathbf{v_i}.$$

Node performance of PH|PH|1 queues

In the case of infinite buffer capacity, the expressions presented for the PH|PH|1|K queue in the previous section can still be applied, provided that the steady-state probability vectors $\mathbf{v_i}$ are available in a form that allows to calculate the, now infinite, sums. For example, if we assume that a matrix-geometric solution method (see Section 3.3) is employed to compute the steady-state probabilities, the vectors $\mathbf{v_i}$ have the so-called matrix geometric form

$$\mathbf{v_i} = \mathbf{v_1}\mathbf{R}^{i-1}, \quad \mathbf{R} \in \mathbb{R}^{lm \times lm}, \quad i = 1, 2, \ldots ,$$

where $\mathbf{R}$ is the entry-wise smallest non-negative solution of the matrix-quadratic equation

$$\mathbf{A}^0\boldsymbol{\alpha} \otimes \boldsymbol{\beta} + \mathbf{R}(\mathbf{A} \oplus \mathbf{B}) + \mathbf{R}^2(\mathbf{I} \otimes \mathbf{B}^0\beta) = \mathbf{0}.$$

The j-th moment of the queue length distribution is then given by

$$\mathrm{E}[N^j] = \sum_{i=1}^{\infty} i^j \mathbf{v_i}\mathbf{1} = \sum_{i=1}^{\infty} i^j \mathbf{v_1}\mathbf{R}^{i-1}\mathbf{1}, \tag{5.17}$$

which yields in case $j = 1$:

$$\mathrm{E}[N] = \mathbf{v_1}(\mathbf{I} - \mathbf{R})^{-2}\mathbf{1}.$$

Similarly, the other node performance measures can be obtained.

5.3.7 Network performance

Many results for the network performance measures developed by Whitt for QNA (see Section 5.2.7) can also be applied to FiFiQueues when respecting the fact that, due to losses at finite queues, the departure rate of a node may differ from the total arrival rate to that node. Additionally, one has to decide how loss traffic streams should be treated in the computation of network wide performance results. For example, the following question has to be answered: should the expected number of visits $\mathrm{E}[V_i]$ also include rejections due to full buffers? As this is only a problem of "interpretation" of the results, we will not discuss it further here.

5.3.8 Complexity

Traffic computation

In FiFiQueues the traffic descriptor of the outgoing traffic depends in a complex, non-linear way on the incoming traffic. Thus, unlike the QNA method, FiFiQueues clearly requires an iterative computation scheme to compute the descriptors of the internal traffic streams. A deeper discussion of FiFiQueues' iteration behavior is given in Section 5.4. Here, we will analyze the complexity of the operations that have to be performed for each node during each iteration.

First, we can safely neglect the traffic merging and splitting steps in our discussion. They only consist of a small number of additions and multiplications. The most time and space consuming operation is the service operation. It can be divided into three phases:

1. fitting of the PH distribution to the arrival traffic,
2. computation of the steady-state probability vector of the underlying CTMC, and

3. computation of the departure traffic descriptor (and, if needed, of the loss traffic descriptor).

Again, we can neglect the first phase since its time complexity is $O(1)$. For the second phase, we differ between finite and infinite queueing stations.

If the queueing capacity is finite, so is the CTMC. Let l be the size of the arrival PH process, i.e., the number of states of its CTMC representation, m the size of the service PH process and K the queueing capacity. Then, the generator matrix is of size $(l + lmK) \times (l + lmK)$. This corresponds to a finite QBD with $N_0 = l$ and $N = lm$ (see Section 3.4). The current implementation of FiFiQueues uses the Gauss-Seidel method for finite capacities[1]. If the descriptor of the loss traffic is required, operations of similar complexity have to be performed to compute the product $\mathbf{v}(-\mathbf{L_0})^{-1}$. For unbounded queueing capacity, the LR algorithm is used (see Section 3.3).

Once the steady-state solution is known, the departure traffic descriptor can be computed. Both for finite and infinite queueing stations, this only requires a small number of matrix vector multiplications. Note that the moments b_i of the service process needed by Equation (5.11) are constant for a given network and hence can be precomputed once.

Node performance and network performance computation

Since the network performance computation is comparable to that of the QNA method, we only discuss the complexity of the node performance computation here.

Concerning finite queueing stations, the computation of the mean and variance of the queue length requires the summation over the $lm(K-1)$ entries of the steady-state probability vector. For the moments of the waiting time, we have to invert matrix $\mathbf{B}$ of size $m \times m$ which can be seen as a constant time operation even for very complex PH representations of the service process (say, $m = 50$).

In case of infinite queueing capacity, the complexity depends on the employed solution method. Assuming a matrix-geometric solution method, the expression $\mathrm{E}[N] = \mathbf{v_1}(\mathbf{I} - \mathbf{R})^{-2}\mathbf{1}$ we gave for the mean queue length in Section 5.3.6, requires the vector $\mathbf{X} = \mathbf{v_1}(\mathbf{I} - \mathbf{R})^{-2}$ which can be obtained by solving the linear system $\mathbf{X}(\mathbf{I} - \mathbf{R})^2 = \mathbf{v_1}$ of order lm.

Conclusion

Summarizing, we can state that the complexity of FiFiQueues is nearly completely dominated by the computation of the steady-state solution for each node. However, for a given network model, the time for the analysis cannot be easily predicted. First,

[1]for merely historical reasons; a new implementation using the Cyclic Reduction method (see Section 3.4.3) is tested in Section 6.2.4.

at the level of the surrounding iteration scheme, the number of required iterations cannot be determined a priori (besides for scenarios with trivial network topologies). Additionally, the complexity of the node analysis is at least quadratic in the number l of phases of the arrival PH representation. This number l directly depends on the variance of the arrival stream and may change from node to node and from iteration to iteration.

5.4 Existence of the fixed point

We have stated in Section 2.3.3 that detailed knowledge about the fixed-point iteration behavior of queueing analyzers is in general not available due to the complexity of the involved algorithms. It is often not known whether the searched fixed point exists, is unique or will be reached. However, some intermediate results are available for FiFiQueues which we will present here. In the following we give a proof that the fixed point exists for a modified version of the original FiFiQueues algorithm.

5.4.1 Notation and Brouwer's theorem

Given a queueing network with n stations, we define $D \subset \mathbb{R}^{2n}$ where the tuple

$$\left(\left\langle \lambda_{a,1}, c_{a,1}^2 \right\rangle, \ldots, \left\langle \lambda_{a,n}, c_{a,n}^2 \right\rangle\right) \in D$$

gives for each node $i \in \{1, \ldots, n\}$ the traffic descriptor $\left\langle \lambda_{a,i}, c_{a,i}^2 \right\rangle$ of its arrival traffic. Then the operations performed by FiFiQueues during step $k+1$ of the fixed-point iteration can be expressed as a function $H : D \to D$ [109] which computes from the traffic descriptor $\mathbf{d}^k$ obtained from step k the new traffic descriptor $\mathbf{d}^{k+1}$, that is,

$$\mathbf{d}^{k+1} = H(\mathbf{d}^k),$$

where $\mathbf{d}^0$ is the initial traffic descriptor used in the iteration. We use the *Brouwer fixed-point theorem* [104] to prove the existence of the fixed point for the function H. It states:

> *Let $D \subset \mathbb{R}^m$ be a non-empty, closed, convex, and bounded set, and $H : D \to D$ continuous. Then H has a fixed point.*

A first proof of the existence of the fixed point has been discussed in [109] for special service processes. The proof presented in the following applies to arbitrary PH renewal service processes. We first show in Section 5.4.2 that the requirements to the set D are met. The continuity of H is shown in Sections 5.4.3–5.4.5.

5.4.2 Properties of D

Lower and upper bounds for the arrival rate $\lambda_{a,i}$ of a node i exist. It holds

$$0 \leq \lambda_{a,i} \leq \lambda_{max,i}.$$

where the upper bound $\lambda_{max,i}$ is the maximum arrival rate that will only be reached if all queueing stations operate with a load of 100%. It is given by

$$\boldsymbol{\lambda}_{\mathbf{max}} = \boldsymbol{\lambda}_{\mathbf{ext}}(\mathbf{I} - \boldsymbol{\Gamma})^{-1},$$

where $\boldsymbol{\Gamma}$ is the routing matrix and $\lambda_{ext,i}$ is the rate of the external traffic arriving to node i. As previously explained, FiFiQueues limits the squared coefficient of variation to $\frac{1}{10}$ to prevent the generation of PH distributions with more than 10 states. Originally, no upper bound is provided for the coefficients but we can safely define

$$c^2_{a,i} := \min(c^2_{max}, c^2_{a,i}), \qquad i = 1, \ldots, n,$$

with $c^2_{max} = 1000$ without affecting the analysis. We thus obtain that D is the non-empty, closed and convex interval

$$\left[\left(\langle 0, \tfrac{1}{10}\rangle, \ldots, \langle 0, \tfrac{1}{10}\rangle\right), \left(\langle \lambda_{max,1}, c^2_{max}\rangle, \ldots, \langle \lambda_{max,1}, c^2_{max}\rangle\right)\right] \subset \mathbb{R}^{2n},$$

as required.

5.4.3 Continuity of H

The function H performs for each node the following operations to compute the traffic descriptors for the next iteration in the algorithm:

1. the service operation;
2. the traffic splitting;
3. the traffic merging.

The traffic merging step is a function of the traffic descriptors generated during the splitting operation. The traffic splitting, in turn, is a function of the departure-traffic descriptor as computed by the service operation. An inspection of the involved terms for the traffic merging (Equation (5.6) and (5.7)), the traffic splitting (Equation (5.8)), and the service operation (Equations (5.10), (5.12), (5.14), and (5.15)) shows that the proof of continuity reduces to the question whether, for a given node, the loss probability π (in case of a finite queue) and the variance σ_0^2 are continuous functions of the arrival traffic $\langle \lambda_a, c_a^2 \rangle$. Since π and σ_0^2 depend on the stationary distribution $\mathbf{v}$ of the underlying CTMC we can make use of the following theorem [78] to prove this continuity:

The stationary distribution of a CTMC as function of the transition rates $\lambda_1, \dots, \lambda_n$ of the generator matrix is continuous for all $\lambda_i > 0, i = 1, \dots, n$ if the CTMC has exactly one irreducible set of states.

The underlying CTMC of a queueing station has exactly one irreducible set of states since it is a QBD. Then, the question is, how do the transition rates of the generator matrix depend on $\langle \lambda_a, c_a^2 \rangle$? FiFiQueues uses the traffic descriptor to determine a PH renewal process that represents the arrival traffic. This arrival PH process is then combined with the service PH process of the node to construct the generator matrix. For fixed c_a^2 the transition rates of the generator matrix are a continuous function of the arrival rate λ_a. The theorem then yields the continuity of $\mathbf{v}$ as function of λ_a. However, varying c_a^2 may cause FiFiQueues to change the size and structure of the PH representation. Such a change also influences the size and structure of the QBD. As consequence, the theorem can only be applied for values of c_a^2 that do not cause such a change. We obtain:

1. $\mathbf{v}$ is continuous for $c_a^2 > 1$, since then the PH distribution always takes the same, hyper-exponential, form.
2. $\mathbf{v}$ is continuous for $c_a^2 \in \left(\frac{1}{m+1}, \frac{1}{m}\right)$, for all $m \in \{1, \dots, 9\}$.

The other cases, i.e., $c_a^2 = \frac{1}{m}, m \in \{1, \dots, 10\}$, have to be separately discussed. In the following we only show the continuity of π. The proof for σ_0^2 is done in a similar way.

5.4.4 Continuity for $c_a^2 = 1$

We show that

$$\lim_{c_a^2 \nearrow 1} \pi(c_a^2) = \pi(c_a^2 = 1) = \lim_{c_a^2 \searrow 1} \pi(c_a^2)$$

which yields the continuity of π around $c_a^2 = 1$.

Case $\mathbf{c_a^2 = 1}$

If $c_a^2 = 1$, the arrival PH distribution is a negative-exponential distribution with rate λ_a. Following the notation used in Section 5.3.5, we obtain the steady-state probability distribution $\mathbf{v}^=$ by solving the global balance equations

$$\left.\begin{array}{rcl} \mathbf{v_0^=}(-\lambda_a) + \mathbf{v_1^=}\mathbf{B^0} & = & 0 \\ \mathbf{v_0^=}\lambda_a\boldsymbol{\beta} + \mathbf{v_1^=}(-\lambda_a\mathbf{I} + \mathbf{B}) + \mathbf{v_2^=}\mathbf{B^0}\boldsymbol{\beta} & = & 0 \\ \mathbf{v_1^=}\lambda_a\mathbf{I} + \mathbf{v_2^=}(-\lambda_a\mathbf{I} + \mathbf{B}) + \mathbf{v_3^=}\mathbf{B^0}\boldsymbol{\beta} & = & 0 \\ \dots & & \\ \mathbf{v_{K-1}^=}\lambda_a\mathbf{I} + \mathbf{v_K^=}\mathbf{B} & = & 0 \end{array}\right\} \qquad (5.18)$$

and

$$\mathbf{v}^{=} \cdot \mathbf{1} = 1,$$

where $(\boldsymbol{\beta}, \mathbf{B})$ is the service PH process and K is the queueing capacity. The loss probability $\pi^{=}$ is then given as

$$\pi^{=} = \pi(c_a^2 = 1) = \frac{1}{\lambda_a} \mathbf{v}_{\mathbf{K}}^{=} (\lambda_a \otimes \mathbf{I}) \cdot \mathbf{1}.$$

Case $\mathbf{c_a^2} \searrow \mathbf{1}$

If $c_a^2 > 1$, FiFiQueues selects the PH renewal process $(\boldsymbol{\alpha}, \mathbf{A})$ as representation of the arrival traffic with

$$\mathbf{A} = \begin{pmatrix} -\lambda_0 & 0 \\ 0 & -\lambda_1 \end{pmatrix} \quad \text{and} \quad \boldsymbol{\alpha} = (p, 1-p),$$

where $p = \frac{1}{2} + \frac{1}{2}\sqrt{\frac{c_a^2 - 1}{c_a^2 + 1}}$, $\lambda_0 = 2p\lambda_a$ and $\lambda_1 = 2(1-p)\lambda_a$.

Let $\mathbf{v}^{>}$ be the steady-state probability distribution of the resulting QBD. To ease the following calculations we split the components $\mathbf{v}_{\mathbf{i}}^{>}$, i$= 0, \ldots, K$, of the probability distribution vector into two parts $\mathbf{v}_{\mathbf{i}}^{>} = (\mathbf{v}_{\mathbf{i1}}^{>}, \mathbf{v}_{\mathbf{i2}}^{>})$ where $\mathbf{v}_{\mathbf{i1}}^{>}$ and $\mathbf{v}_{\mathbf{i2}}^{>}$ are associated with the first resp. the second state of the arrival PH process. The vector $\mathbf{v}^{>}$ is then determined by the following equations:

$$\begin{aligned}
\mathbf{v}_{\mathbf{01}}^{>}(-\lambda_0) + \mathbf{v}_{\mathbf{11}}^{>}\mathbf{B}^{\mathbf{0}} &= 0, && (5.19)\\
\mathbf{v}_{\mathbf{02}}^{>}(-\lambda_1) + \mathbf{v}_{\mathbf{12}}^{>}\mathbf{B}^{\mathbf{0}} &= 0, && (5.20)\\
\mathbf{v}_{\mathbf{01}}^{>}p\lambda_0\boldsymbol{\beta} + \mathbf{v}_{\mathbf{02}}^{>}p\lambda_1\boldsymbol{\beta} + \mathbf{v}_{\mathbf{11}}^{>}(-\lambda_0\mathbf{I} + \mathbf{B}) + \mathbf{v}_{\mathbf{21}}^{>}\mathbf{B}^{\mathbf{0}}\boldsymbol{\beta} &= 0, && (5.21)\\
\mathbf{v}_{\mathbf{01}}^{>}(1-p)\lambda_0\boldsymbol{\beta} + \mathbf{v}_{\mathbf{02}}^{>}(1-p)\lambda_1\boldsymbol{\beta} + \mathbf{v}_{\mathbf{12}}^{>}(-\lambda_1\mathbf{I} + \mathbf{B}) + \mathbf{v}_{\mathbf{22}}^{>}\mathbf{B}^{\mathbf{0}}\boldsymbol{\beta} &= 0, && (5.22)\\
\mathbf{v}_{\mathbf{11}}^{>}p\lambda_0\mathbf{I} + \mathbf{v}_{\mathbf{12}}^{>}p\lambda_1\mathbf{I} + \mathbf{v}_{\mathbf{21}}^{>}(-\lambda_0\mathbf{I} + \mathbf{B}) + \mathbf{v}_{\mathbf{31}}^{>}\mathbf{B}^{\mathbf{0}}\boldsymbol{\beta} &= 0, && (5.23)\\
\mathbf{v}_{\mathbf{11}}^{>}(1-p)\lambda_0\mathbf{I} + \mathbf{v}_{\mathbf{12}}^{>}(1-p)\lambda_1\mathbf{I} + \mathbf{v}_{\mathbf{22}}^{>}(-\lambda_1\mathbf{I} + \mathbf{B}) + \mathbf{v}_{\mathbf{32}}^{>}\mathbf{B}^{\mathbf{0}}\boldsymbol{\beta} &= 0, && (5.24)\\
&\cdots\\
\mathbf{v}_{\mathbf{K-1,1}}^{>}p\lambda_0\mathbf{I} + \mathbf{v}_{\mathbf{K-1,2}}^{>}p\lambda_1\mathbf{I} + \mathbf{v}_{\mathbf{K1}}^{>}((p-1)\lambda_0\mathbf{I} + \mathbf{B}) + \mathbf{v}_{\mathbf{K2}}^{>}p\lambda_1\mathbf{I} &= 0, && (5.25)\\
\mathbf{v}_{\mathbf{K-1,1}}^{>}(1-p)\lambda_0\mathbf{I} + \mathbf{v}_{\mathbf{K-1,2}}^{>}(1-p)\lambda_1\mathbf{I} + &\\
\mathbf{v}_{\mathbf{K1}}^{>}(1-p)\lambda_0\mathbf{I} + \mathbf{v}_{\mathbf{K2}}^{>}(-p\lambda_1\mathbf{I} + \mathbf{B}) &= 0, && (5.26)
\end{aligned}$$

and

$$\mathbf{v}^{>} \cdot \mathbf{1} = 1.$$

The loss probability $\pi^{>}$ of the station is then given as

$$\pi^{>} = \frac{1}{\lambda_a}\mathbf{v}_{\mathbf{K}}^{>}(\mathbf{A}^{\mathbf{0}} \otimes \mathbf{I}) \cdot \mathbf{1} = \frac{1}{\lambda_a}(\mathbf{v}_{\mathbf{K1}}^{>}\lambda_0 + \mathbf{v}_{\mathbf{K2}}^{>}\lambda_1)\mathbf{I} \cdot \mathbf{1}. \tag{5.27}$$

Summing Equations (5.19) and (5.20), (5.21) and (5.22), ..., gives:

$$\left.\begin{array}{rcl} s_0 + t_1\mathbf{B^0} &=& 0 \\ -s_0\boldsymbol{\beta} + s_1\mathbf{I} + t_1\mathbf{B} + t_2\mathbf{B^0}\boldsymbol{\beta} &=& 0 \\ -s_1\mathbf{I} + s_2\mathbf{I} + t_2\mathbf{B} + t_3\mathbf{B^0}\boldsymbol{\beta} &=& 0 \\ & \cdots & \\ -s_{K-1}\mathbf{I} + t_K\mathbf{B} &=& 0 \end{array}\right\} \tag{5.28}$$

where $s_i = \mathbf{v^>_{i1}}(-\lambda_0) + \mathbf{v^>_{i2}}(-\lambda_1)$ and $t_i = \mathbf{v^>_{i1}} + \mathbf{v^>_{i2}}$. For $c_a^2 \searrow 1$ we have $p \to \frac{1}{2}$. From this, it follows

$$\lim_{c_a^2 \searrow 1} \lambda_0 = \lim_{c_a^2 \searrow 1} \lambda_1 = \lambda_a,$$

and

$$\lim_{c_a^2 \searrow 1} s_i = -\lambda_a \lim_{c_a^2 \searrow 1} t_i, \qquad i = 0, \ldots, k.$$

By applying these limits to Equation (5.28) we observe their correspondence with Equation (5.18). Hence we, obtain for $c_a^2 \searrow 1$:

$$\lim_{c_a^2 \searrow 1} t_i = \lim_{c_a^2 \searrow 1} (\mathbf{v^>_{i1}} + \mathbf{v^>_{i2}}) = \mathbf{v^=_i},$$

which provides, with Equation (5.27), the desired relationship

$$\lim_{c_a^2 \searrow 1} \pi^> = \pi^=.$$

Case $\mathbf{c_a^2} \nearrow \mathbf{1}$

If $0.5 < c_a^2 < 1$, the arrival PH distribution is a modified hypo-exponential distribution $(\boldsymbol{\alpha}, \mathbf{A})$ as defined in Section 5.3.5 where

$$\mathbf{A} = \begin{pmatrix} -\lambda_0 & \lambda_0 \\ 0 & -\lambda_1 \end{pmatrix} \quad \text{and} \quad \boldsymbol{\alpha} = (1, 0).$$

Let $\mathbf{v}^<$ be the steady-state probability distribution of the resulting QBD. Again, we split the components $\mathbf{v^<_i}$, i= $0, \ldots, K$, of the probability distribution vector into two parts $\mathbf{v^<_i} = (\mathbf{v^<_{i1}}, \mathbf{v^<_{i2}})$, where $\mathbf{v^<_{i1}}$ and $\mathbf{v^<_{i2}}$ are associated with the first resp. the second state of the arrival PH distribution. The vector $\mathbf{v}^<$ is then determined by the following equations:

$$\mathbf{v^<_{01}}(-\lambda_0) + \mathbf{v^<_{11}}\mathbf{B^0} = 0, \tag{5.29}$$

$$\mathbf{v^<_{01}}\lambda_0 + \mathbf{v^<_{02}}(-\lambda_1) + \mathbf{v^<_{12}}\mathbf{B^0} = 0, \tag{5.30}$$

$$\mathbf{v^<_{02}}\lambda_1\boldsymbol{\beta} + \mathbf{v^<_{11}}(-\lambda_0\mathbf{I} + \mathbf{B}) + \mathbf{v^<_{21}}\mathbf{B^0}\boldsymbol{\beta} = 0, \tag{5.31}$$

$$\mathbf{v^<_{11}}\lambda_0\mathbf{I} + \mathbf{v^<_{12}}(-\lambda_1\mathbf{I} + \mathbf{B}) + \mathbf{v^<_{22}}\mathbf{B^0}\boldsymbol{\beta} = 0, \tag{5.32}$$

$$\mathbf{v^<_{12}}\lambda_1\mathbf{I} + \mathbf{v^<_{21}}(-\lambda_0\mathbf{I} + \mathbf{B}) + \mathbf{v^<_{31}}\mathbf{B^0}\boldsymbol{\beta} = 0, \tag{5.33}$$

$$\mathbf{v}^{<}_{\mathbf{21}}\lambda_0\mathbf{I} + \mathbf{v}^{<}_{\mathbf{22}}(-\lambda_1\mathbf{I} + \mathbf{B}) + \mathbf{v}^{<}_{\mathbf{32}}\mathbf{B^0}\boldsymbol{\beta} = 0, \tag{5.34}$$
$$\cdots$$
$$\mathbf{v}^{<}_{\mathbf{K-1,2}}\lambda_1\mathbf{I} + \mathbf{v}^{<}_{\mathbf{K1}}(-\lambda_0\mathbf{I} + \mathbf{B}) + \mathbf{v}^{<}_{\mathbf{K2}}\lambda_1\mathbf{I} = 0, \tag{5.35}$$
$$\mathbf{v}^{<}_{\mathbf{K1}}\lambda_0\mathbf{I} + \mathbf{v}^{<}_{\mathbf{K2}}(-\lambda_1\mathbf{I} + \mathbf{B}) = 0, \tag{5.36}$$

and

$$\mathbf{v}^{<} \cdot \mathbf{1} = 1.$$

The loss probability $\pi^{<}$ of the station is then given as

$$\pi^{<} = \frac{1}{\lambda_a}\mathbf{v}^{<}_{\mathbf{K}}(\mathbf{A^0} \otimes \mathbf{I}) \cdot \mathbf{1} = \frac{1}{\lambda_a}\mathbf{v}^{<}_{\mathbf{K2}}\lambda_1\mathbf{I} \cdot \mathbf{1}. \tag{5.37}$$

From Equation (5.36) we obtain

$$\mathbf{v}^{<}_{\mathbf{K2}}\lambda_1\mathbf{I} = \mathbf{v}^{<}_{\mathbf{K1}}\lambda_0\mathbf{I} + \mathbf{v}^{<}_{\mathbf{K2}}\mathbf{B},$$

which yields with Equation (5.37):

$$\pi^{<} = \frac{1}{\lambda_a}(\mathbf{v}^{<}_{\mathbf{K1}}\lambda_0\mathbf{I} + \mathbf{v}^{<}_{\mathbf{K2}}\mathbf{B}) \cdot \mathbf{1}. \tag{5.38}$$

Using simple substitutions we derive from Equations (5.29)–(5.36):

$$\left.\begin{array}{rcl}
\mathbf{v}^{<}_{\mathbf{01}}(-\lambda_0) + \mathbf{v}^{<}_{\mathbf{11}}\mathbf{B^0} & = & 0 \\
(\mathbf{v}^{<}_{\mathbf{01}}\lambda_0 + \mathbf{v}^{<}_{\mathbf{12}}\mathbf{B^0})\boldsymbol{\beta} + \mathbf{v}^{<}_{\mathbf{11}}(-\lambda_0\mathbf{I} + \mathbf{B}) + \mathbf{v}^{<}_{\mathbf{21}}\mathbf{B^0}\boldsymbol{\beta} & = & 0 \\
(\mathbf{v}^{<}_{\mathbf{11}}\lambda_0\mathbf{I} + \mathbf{v}^{<}_{\mathbf{12}}\mathbf{B} + \mathbf{v}^{<}_{\mathbf{22}}\mathbf{B^0}\boldsymbol{\beta}) + \mathbf{v}^{<}_{\mathbf{21}}(-\lambda_0\mathbf{I} + \mathbf{B}) + \mathbf{v}^{<}_{\mathbf{31}}\mathbf{B^0}\boldsymbol{\beta} & = & 0 \\
\cdots & & \\
(\mathbf{v}^{<}_{\mathbf{K-1,1}}\lambda_0\mathbf{I} + \mathbf{v}^{<}_{\mathbf{K-1,2}}\mathbf{B} + \mathbf{v}^{<}_{\mathbf{K2}}\mathbf{B^0}\boldsymbol{\beta}) + \mathbf{v}^{<}_{\mathbf{K1}}\mathbf{B} + \mathbf{v}^{<}_{\mathbf{K2}}\mathbf{B} & = & 0
\end{array}\right\} \tag{5.39}$$

For $c_a^2 \nearrow 1$ we have $\lambda_0 \to \lambda_a$ and $\lambda_1 \to \infty$. Solving Equations (5.30), (5.32), ..., (5.36) for $\mathbf{v}^{<}_{\mathbf{i2}}$ then gives

$$\mathbf{v}^{<}_{\mathbf{i2}} \to \mathbf{0}, \quad i = 0, \ldots, K.$$

By applying these limits to Equation (5.39) we observe their correspondence with Equation (5.18). Hence we obtain for $c_a^2 \nearrow 1$:

$$\mathbf{v}^{<}_{\mathbf{i1}} \to \mathbf{v}^{=}_{\mathbf{i}} \quad \text{and} \quad \mathbf{v}^{<}_{\mathbf{i2}} \to 0, \qquad i = 0, \ldots, K,$$

which gives, applied to Equation (5.38):

$$\lim_{c_a^2 \nearrow 1} \pi^{<} = \pi^{=},$$

as required.

5.4.5 Continuity for $c_a^2 = \frac{1}{m}, m \in \{2, \dots, 10\}$

The transition rates of the hypo-exponential PH distributions for $c_a^2 < 1$ are defined as functions of the number of phases $m = \left\lceil \frac{1}{c_a^2} \right\rceil$. The inherent discontinuity suggests that the steady-state distribution of the resulting QBD is discontinuous, too. To improve the behavior of the arrival distribution with regard to the continuity, [109] proposes to modify FiFiQueues' PH fitting procedure for $c_a^2 < 1$ as follows. Given the mean inter-arrival time $\mathrm{E}[X] = 1/\lambda_a$ and the squared coefficient of variation c_a^2, we fit the PH distribution $(\boldsymbol{\alpha}, \mathbf{A})$ with $m = \lceil \frac{1}{c^2} \rceil$ phases and initial probability vector $\boldsymbol{\alpha} = (1, 0, \dots, 0)$. The matrix $\mathbf{A}$ is given as

$$\mathbf{A} = \begin{pmatrix} -\lambda_0 & \lambda_0 & & & \\ & -\lambda_1 & \lambda_1 & & \\ & & \ddots & \ddots & \\ & & & -\lambda_{m-2} & \lambda_{m-2} \\ & & & & -\lambda_{m-1} \end{pmatrix}, \tag{5.40}$$

where

$$\begin{aligned} \lambda_i &= 1/(c_a^2 \mathrm{E}[X]), \text{ for } 0 \leq i < m-2, \\ \lambda_{m-1} &= \frac{1 - (m-2)c_a^2 + \sqrt{(c_a^2)^2(2m - m^2) + 2c_a^2(m-1) - 1}}{\mathrm{E}[X]((m-1)(m-2)(c_a^2)^2 + c_a^2(3-2m) + 1)}, \\ \lambda_{m-2} &= \frac{\lambda_{m-1}}{\mathrm{E}[X]\lambda_{m-1}(1 - (m-2)c_a^2) - 1} \end{aligned}$$

As can be seen, the transition rates λ_i for $i < m-2$ are now continuous as function of c_a^2. This causes that important statistics of this new PH renewal process, e.g., the third moment of the inter-arrival time, are now continuous at values of c_a^2 where a size change happens. Note that this not true for the original PH distribution. Figure 5.1 illustrates this by comparing the third moment of both PH distributions for values of c_a^2 around 0.5 (i.e., when the size m changes from 3 to 2).

For $c_a^2 \nearrow \frac{1}{m}$, $m \in \{1, \dots, 10\}$, the new PH distribution yields

$$\lambda_i \longrightarrow m\lambda_a, \qquad \text{for } 0 \leq i < m-1, \tag{5.41}$$

$$\lambda_{m-1} \longrightarrow \infty. \tag{5.42}$$

Hence, the proof of $\lim_{c_a^2 \nearrow 1} \pi(c_a^2) = \pi(c_a^2 = 1)$ for $m = 1$, as given in the previous section, is still valid and we only have to prove that

$$\lim_{c_a^2 \nearrow \frac{1}{m}} \pi(c_a^2) = \pi(c_a^2 = \frac{1}{m})$$

for $m \in \{2, \dots, 10\}$. For $c_a^2 = \frac{1}{m}$ the arrival PH distribution $(\boldsymbol{\alpha}_=, \mathbf{A}_=)$ is an Erlang-m distribution. Again, let $\mathbf{v}^=$ be the steady-state probability vector of the resulting

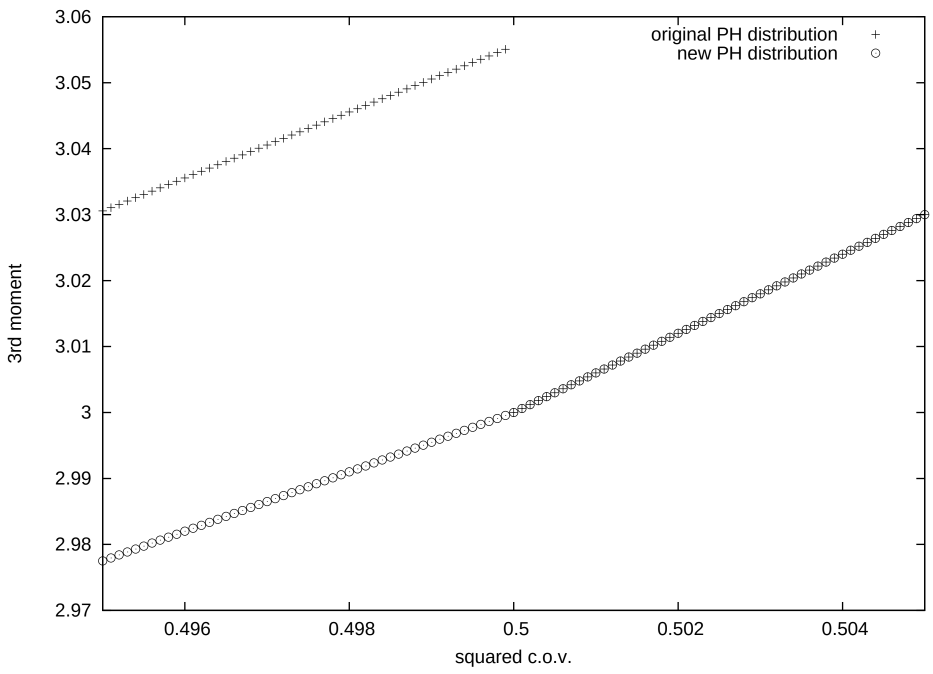


Figure 5.1: Third moment of the original (Equation (5.9)) and the modified (Equation (5.40)) PH distribution as function of c_a^2

QBD with $c_a^2 = \frac{1}{m}$, $m \in \{2, \dots, 10\}$. We split the components $\mathbf{v_i^=}$, i= $0, \dots, K$, of the probability distribution vector into m parts $\mathbf{v_i^=} = (\mathbf{v_{i1}^=}, \dots, \mathbf{v_{im}^=})$ where $\mathbf{v_{ij}^=}$ is associated with the j-th state of the arrival PH distribution. The probabilities are determined by $\mathbf{v}^= \cdot \mathbf{1} = 1$ and by the global balance equations of the QBD. For level 0 of the QBD, when no customers are in the queueing station, we obtain

$$\begin{aligned}
\mathbf{v_{01}^=}(-\lambda^=) + \mathbf{v_{11}^=}\mathbf{B^0} &= 0, \\
\mathbf{v_{01}^=}\lambda^= + \mathbf{v_{02}^=}(-\lambda^=) + \mathbf{v_{12}^=}\mathbf{B^0} &= 0, \\
&\dots \\
\mathbf{v_{0,m-1}^=}\lambda^= + \mathbf{v_{0m}^=}(-\lambda^=) + \mathbf{v_{1m}^=}\mathbf{B^0} &= 0,
\end{aligned}$$

where $\lambda^= = m\lambda_a$. For level $1 \le i < K$, we have

$$\begin{aligned}
\mathbf{v_{i-1,m}^=}\lambda^=\boldsymbol{\beta} + \mathbf{v_{i1}^=}(-\lambda^=\mathbf{I} + \mathbf{B}) + \mathbf{v_{i+1,1}^=}\mathbf{B^0}\boldsymbol{\beta} &= 0, \\
\mathbf{v_{i1}^=}\lambda^=\mathbf{I} + \mathbf{v_{i2}^=}(-\lambda^=\mathbf{I} + \mathbf{B}) + \mathbf{v_{i+1,2}^=}\mathbf{B^0}\boldsymbol{\beta} &= 0, \\
&\dots \\
\mathbf{v_{i,m-1}^=}\lambda^=\mathbf{I} + \mathbf{v_{im}^=}(-\lambda^=\mathbf{I} + \mathbf{B}) + \mathbf{v_{i+1,m}^=}\mathbf{B^0}\boldsymbol{\beta} &= 0,
\end{aligned}$$

and, finally, for level K:

$$\mathbf{v_{K-1,m}^=}\lambda^=\mathbf{I} + \mathbf{v_{K1}^=}(-\lambda^=\mathbf{I} + \mathbf{B}) + \mathbf{v_{Km}^=}\lambda_{m-1}\mathbf{I} = 0,$$

$$\begin{aligned}
\mathbf{v^{=}_{K1}}\lambda^{=}\mathbf{I} + \mathbf{v^{=}_{K2}}(-\lambda^{=}\mathbf{I} + \mathbf{B}) &= 0,\\
&\cdots\\
\mathbf{v^{=}_{K,m-1}}\lambda^{=}\mathbf{I} + \mathbf{v^{=}_{Km}}(-\lambda^{=}\mathbf{I} + \mathbf{B}) &= 0.
\end{aligned}$$

The loss probability $\pi^{=}$ of the station is given by

$$\pi^{=} = \frac{1}{\lambda_a}\mathbf{v^{=}_{K}}(\mathbf{A^0_{=}} \otimes \mathbf{I}) \cdot \mathbf{1} = \frac{1}{\lambda_a}\mathbf{v^{=}_{Km}}\lambda^{=}\mathbf{I} \cdot \mathbf{1}.$$

For $\frac{1}{m+1} \leq c_a^2 < \frac{1}{m}$ the resulting arrival PH process $(\boldsymbol{\alpha}_{<}, \mathbf{A}_{<})$ has $m+1$ states and the steady-state probability vector $\mathbf{v}^{<}$ of the QBD has additional components $\mathbf{v^{<}_{i,m+1}}, i = 0, \ldots, K$. Let $\lambda_0, \ldots, \lambda_{m+1}$ be the transition rates of the PH process. The global balance equations for level 0 are

$$\begin{aligned}
\mathbf{v^{<}_{01}}(-\lambda_0) + \mathbf{v^{<}_{11}}\mathbf{B^0} &= 0,\\
\mathbf{v^{<}_{01}}\lambda_0 + \mathbf{v^{<}_{02}}(-\lambda_1) + \mathbf{v^{<}_{12}}\mathbf{B^0} &= 0,\\
&\cdots\\
\mathbf{v^{<}_{0m}}\lambda_{m-1} + \mathbf{v^{<}_{0,m+1}}(-\lambda_m) + \mathbf{v^{<}_{1,m+1}}\mathbf{B^0} &= 0.
\end{aligned}$$

For level $1 \leq i < K$, we have

$$\begin{aligned}
\mathbf{v^{<}_{i-1,m+1}}\lambda_m\boldsymbol{\beta} + \mathbf{v^{<}_{i1}}(-\lambda_0\mathbf{I} + \mathbf{B}) + \mathbf{v^{<}_{i+1,1}}\mathbf{B^0}\boldsymbol{\beta} &= 0,\\
\mathbf{v^{<}_{i1}}\lambda_0\mathbf{I} + \mathbf{v^{<}_{i2}}(-\lambda_1\mathbf{I} + \mathbf{B}) + \mathbf{v^{<}_{i+1,2}}\mathbf{B^0}\boldsymbol{\beta} &= 0,\\
&\cdots\\
\mathbf{v^{<}_{im}}\lambda_{m-1}\mathbf{I} + \mathbf{v^{<}_{i,m+1}}(-\lambda_m\mathbf{I} + \mathbf{B}) + \mathbf{v^{<}_{i+1,m+1}}\mathbf{B^0}\boldsymbol{\beta} &= 0,
\end{aligned}$$

and for level K:

$$\begin{aligned}
\mathbf{v^{<}_{K-1,m+1}}\lambda_m\mathbf{I} + \mathbf{v^{<}_{K1}}(-\lambda_0\mathbf{I} + \mathbf{B}) + \mathbf{v^{<}_{K,m+1}}\lambda_m\mathbf{I} &= 0 \qquad (5.43)\\
\mathbf{v^{<}_{K1}}\lambda_0\mathbf{I} + \mathbf{v^{<}_{K2}}(-\lambda_1\mathbf{I} + \mathbf{B}) &= 0 \qquad (5.44)\\
&\cdots\\
\mathbf{v^{<}_{Km}}\lambda_{m-1}\mathbf{I} + \mathbf{v^{<}_{K,m+1}}(-\lambda_m\mathbf{I} + \mathbf{B}) &= 0. \qquad (5.45)
\end{aligned}$$

The loss probability $\pi^{<}$ is given by

$$\pi^{<} = \frac{1}{\lambda_a}\mathbf{v^{<}_{K}}(\mathbf{A^0_{<}} \otimes \mathbf{I}) \cdot \mathbf{1} = \frac{1}{\lambda_a}\mathbf{v^{<}_{K,m+1}}\lambda_m\mathbf{I} \cdot \mathbf{1}. \qquad (5.46)$$

From Equation (5.45) we obtain

$$\mathbf{v^{<}_{K,m+1}}\lambda_m\mathbf{I} = \mathbf{v^{<}_{Km}}\lambda_{m-1}\mathbf{I} + \mathbf{v^{<}_{K,m+1}}\mathbf{B},$$

which yields with Equation (5.46):

$$\pi^{<} = \frac{1}{\lambda_a}(\mathbf{v^{<}_{Km}}\lambda_{m-1}\mathbf{I} + \mathbf{v^{<}_{K,m+1}}\mathbf{B}) \cdot \mathbf{1}. \qquad (5.47)$$

Solving the global balance equations for $\mathbf{v}^{<}_{\mathbf{i,m+1}}$ and applying the limits from (5.41) and (5.42) gives

$$\mathbf{v}^{<}_{\mathbf{i,m+1}} \to \mathbf{0}, \quad i = 0, \ldots, K.$$

Similar to the case $c_a^2 \nearrow 1$ of the original PH distribution (see Equation (5.39)), transforming the global balance equations finally yields

$$\begin{aligned} \mathbf{v}^{<}_{\mathbf{ij}} &\to \mathbf{v}^{=}_{\mathbf{ij}}, \\ \mathbf{v}^{<}_{\mathbf{i,m+1}} &\to 0, \qquad\qquad i = 0, \ldots, K \text{ and } j = 1, \ldots, m, \end{aligned}$$

which gives, applied to Equation (5.47):

$$\lim_{c_a^2 \nearrow \frac{1}{m}} \pi^{<} = \pi^{=},$$

as required.

We now have shown that the loss probability π is a continuous function of c_a^2. Together with the continuity of the variance σ_0^2 (not shown here), this yields the continuity of the the function H, and, as consequence, the existence of the fixed point.

5.5 Traffic descriptors with m moments

In the previous sections, we have seen that QNA and FiFiQueues only use the first and the second moment of the inter-arrival time distribution as traffic descriptor. So, it is natural to ask whether the quality of the analysis can be improved if more information about the arrival time distribution is included in the traffic descriptor. This can be in done in several ways and we will discuss the most obvious approach in this section: we try to extend FiFiQueues by traffic descriptors containing three or more moments, encouraged by Equation (5.11) which allows to efficiently compute any desired moment of the inter-departure time distribution of a node. However, note that the number of moments a "good" traffic descriptor should contain is very difficult to determine and may vary from model to model. The intensive research in the context of approximations of GI|M|· queues over the last twenty years showed that in some cases, two-moment approximations are adequate, in other cases three-moment approximations are adequate, and in still other cases, good results can only be obtained if other properties like autocorrelation are taken into account [57].

In order to avoid repetition, we deviate from the structure of the previous descriptions, and rather discuss the extensions and changes to FiFiQueues. The model class as well as the network performance computation are not repeated here. Instead, we directly start with the description of the new traffic descriptor and continue with the superposition and merging of traffic streams. Concerning the service operation, the PH-fitting step now becomes so complex that we have devoted a separate section to it.

5.5.1 The traffic descriptor

The traffic descriptor consists of the first m moments of the inter-arrival time distribution:

$$\langle a_1, \ldots, a_m \rangle,$$

where a_i is the i-th non-central moment. The same description is also chosen for the external arrival processes. Thus, FiFiQueues can be seen as the special case with $m = 2$.

5.5.2 Superposition of traffic streams

Given two incoming streams described by $\langle x_1, \ldots, x_m \rangle$ and $\langle y_1, \ldots, y_m \rangle$, we want to compute the traffic descriptor $\langle r_1, \ldots, r_m \rangle$ of the resulting stream. Let $\lambda_X = 1/x_1$, $\lambda_Y = 1/y_1$ and $\lambda_R = 1/r_1$.

If the two incoming streams are generated by renewal processes, the resulting traffic descriptor can be exactly computed by fitting the two incoming streams to PH renewal processes (see Section 5.5.5). If $(\boldsymbol{\alpha}, \mathbf{A})$ and $(\boldsymbol{\beta}, \mathbf{B})$ are the corresponding phase-type representations, their superposition can be expressed by the MAP

$$(\mathbf{A} \oplus \mathbf{B}, \mathbf{A^0}\boldsymbol{\alpha} \oplus \mathbf{B^0}\boldsymbol{\beta}),$$

which yields the searched moments. This approach corresponds to what is termed the stationary-interval method in QNA.

Inside a queueing network the renewal assumption is wrong in most cases. In order to respect correlations in the traffic streams, Whitt complemented the stationary-interval method with the asymptotic method and obtained the hybrid formula for the second moment (see Section 5.2.3). It is, however, an open research question how the hybrid approach could be extended to higher moments, especially to the third moment or the *skewness*. Similar to the squared coefficient of variation, the skewness γ of a traffic stream with inter-arrival time X is defined as

$$\gamma = \frac{\mathrm{E}[(X - \mathrm{E}[X])^3]}{\mathrm{Var}[X]^{3/2}} = \frac{\mathrm{E}[X^3] - 3\mathrm{E}[X]\mathrm{E}[X^2] + 2\mathrm{E}[X]^3}{(\mathrm{E}[X^2] - \mathrm{E}[X]^2)^{3/2}}.$$

We continue to use Whitt's hybrid approximation of the squared coefficient of variation and choose for the skewness γ_R of the superposition:

$$\gamma_R = \gamma_{SI},$$

where γ_{SI} is the skewness as computed by the stationary-interval method.

5.5.3 Splitting traffic streams

Given a stream $\langle x_1, \ldots, x_m \rangle$ and the splitting probability p, we search the traffic descriptor $\langle s_1, \ldots, s_m \rangle$ of the splitted stream.

Let $X_1, X_2, \ldots$ be a sequence of successive inter-arrival times of the original traffic stream. Again, we assume that the original stream is a renewal process, hence the X_i are independent and identically distributed. The inter-arrival time S of the splitted stream is the random sum $S = \sum_{i=1}^{N} X_i$ where N is a discrete random variable with geometric distribution $P(N = n) = p(1-p)^{n-1}$. Random sums of this form are well known and we obtain as generating function of S [91]:

$$G_S(z) = \frac{pG_X(z)}{1-(1-p)G_X(z)},$$

where $G_X(z)$ is the generating function of the original inter-arrival time X. For the i-th derivative $G_Y^{(i)}$ of the generating function G_Y of a random variable Y it holds

$$G_Y^{(i)}(1) = \mathrm{E}\left[\frac{Y!}{(Y-i)!}\right].$$

From this follows

$$\begin{aligned} G_Y(1) &= 1, \\ G_Y'(1) &= \mathrm{E}[Y], \\ G_Y''(1) &= \mathrm{E}[Y^2] - \mathrm{E}[Y], \\ G_Y'''(1) &= \mathrm{E}[Y^3] - 3\mathrm{E}[Y^2] + 2\mathrm{E}[Y], \\ &\ldots \end{aligned}$$

We can derive for $G_S(z)$:

$$\begin{aligned} s_1 &= \frac{1}{p}x_1, \\ s_2 &= \frac{1}{p^2}\left(2(1-p)x_1^2 + px_2\right), \\ s_3 &= \frac{1}{p^3}\left(6(1-p)^2x_1^3 + 6p(1-p)x_1x_2 + p^2x_3\right), \\ &\ldots \end{aligned}$$

One can easily verify that the expressions for s_1 and s_2 conform with the splitting formulae of QNA (see Section 5.2.4).

5.5.4 The service operation and the node performance

Before we discuss the PH-fitting step of the service operation, we state that the core of the node analysis of FiFiQueues, namely the analysis of the underlying QBD, is not much affected by the new traffic descriptor — provided that the PH representation of the arrival process is available — and we can still use Equation (5.11) to compute any moment of the departure process. For example, the third moment is

$$d_3 = b_3 + \mathbf{v_{D,0}}(-3\mathbf{A}^{-1}\mathbf{1}b_2 + 6\mathbf{A}^{-2}\mathbf{1}b_1 - 6\mathbf{A}^{-3}\mathbf{1}).$$

The moments of the other performance measures can be deduced from the corresponding equations; see Equations (5.13), (5.17), resp. (5.16) for the inter-loss time, the queue length for PH|PH|1|K queues, resp. the queue length for PH|PH|1 queues. Moments of the waiting time distribution can be derived from [13, Eq. (4.4)].

We now move to the critical part of the service operation: the fitting of a PH distribution to the incoming traffic described by $\langle a_1, \dots, a_m \rangle$.

5.5.5 PH fitting for m moments

In the context of moment fitting procedures, it is appropriate to introduce the second standardized moment c (the coefficient of variation) defined by

$$c = \bar{a}_2^{1/2}/a_1,$$

and the i-th standardized moment s_i defined for $i > 2$ by

$$s_i = \bar{a}_i/\bar{a}_2^{i/2},$$

where $\bar{a}_i$ is the i-th centralized moment. The third standardized moment is the skewness γ already introduced in Section 5.5.2. Standardized moments are independent of the mean, so fitting procedures for m moments only need to fit moments a_2 through a_m. The correct mean is then obtained by adjusting the scale of the selected distribution.

The original fitting procedure of FiFiQueues only fits against the first and second moment and cannot easily be extended to higher moments. In the following, we briefly describe four fitting procedures for more than two moments.

Moment matching for $m = 3$ with an Erlang-mixture

Johnson and Taaffe presented in [58] a matching procedure for three moments. Unlike older procedures, like the one proposed in [114] which requires $c^2 > 1$, no restrictions on the moments were made (beside $c > 0$ and $\gamma \geq c - 1/c$ which follow from the fact that PH distributions can only represent distribution functions with finite moments).

In [58], the aimed-at PH distribution is a mixture of two Erlang distributions of common order. So, four parameters have to be determined: the common order n, the transition rates λ_1 and λ_2 of the first resp. second Erlang distribution, and the mixing ratio p. For the common order, any $n \geq n^\star$ can be selected where $n^\star$ is the smallest integer that satisfies

$$\begin{aligned} n^\star &> 1/c^2, \quad \text{and} \\ n^\star &> \frac{-\gamma + 1/c^3 + 1/c + 2c}{\gamma - (c - 1/c)}. \end{aligned}$$

The rates and the ratio are given by

$$\begin{aligned} \lambda_1 &= \frac{2A}{-B + \sqrt{B^2 - 4AC}}, \\ \lambda_2 &= \frac{2A}{-B - \sqrt{B^2 - 4AC}}, \\ p &= \lambda_1 \frac{\lambda_2 a_1/n - 1}{\lambda_2 - \lambda_1}, \end{aligned}$$

where

$$\begin{aligned} A &= n(n+2)a_1 y, \\ B &= -(nx + \frac{n(n+2)}{n+1}y^2 + (n+2)a_1^2 y), \\ C &= a_1 x, \\ x &= a_1 a_3 - \frac{n+2}{n+1}a_2^2 \qquad \text{and} \qquad y = a_2 - \frac{n+1}{n}a_1^2. \end{aligned}$$

There are some important remarks to be made regarding this method:

- Choosing the smallest possible order $n^\star$ may result in extreme values for higher moments or in density functions with sharp spikes. Small increments to n (even by 1) can significantly improve the properties of the result.
- Since the resulting PH distribution $(\tau, \mathbf{T})$ is a mixture of two Erlang-distributions, τ and $\mathbf{T}$ obey a special structure that can be exploited to optimize the computations in the node analysis.
- The size of the resulting PH distribution is not always minimal.
- The value of n can become very large and has to be limited. In some cases, it may be necessary to ignore the third moment and to return to a two-moment fitting procedure.

Moment matching for $m = 3$ with an Erlang-Coxian (EC) distribution

Another moment-matching algorithm for $m = 3$ is presented in [87]. Here, the aimed-at distribution is an N-phase Erlang distribution followed by a two-phase

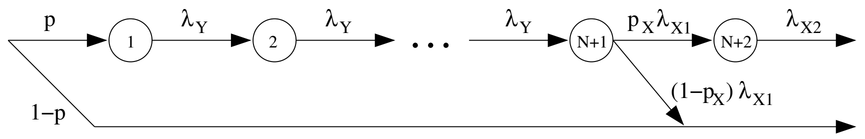


Figure 5.2: The underlying Markov chain of the Erlang-Coxian distribution

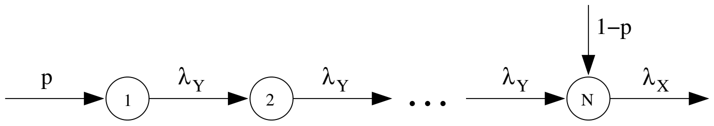


Figure 5.3: The underlying Markov chain of the Erlang-Exp distribution

Coxian distribution. The underlying Markov chain is shown in Figure 5.2. The authors provide three closed-form solutions for the parameters N, p, λ_Y, λ_{X1}, λ_{X_2}, and p_X of the distribution:

1. A simple solution that covers nearly all valid values of $a_1, \ldots, a_3$. It requires at most two phases more than the optimal solution.

2. An improved solution which is defined for all valid values of $a_1, \ldots, a_3$, and uses at most one phase more than the optimal solution. This solution is lacking numerical stability.

3. A numerically stable solution, again defined for all valid values. This solution is stable and uses at most two phases more than the optimal solution.

Moment matching for $m = 3$ with Minimal Acyclic PH (MinAPH) distributions

In [12], Bobbio *et al.* extend the results presented in [87]. They provide exact moment bounds for the first three moments of the Acyclic PH distribution of order N without probability mass at 0 (denoted APH) and with probability mass at 0 (APHM). In addition, they provide an explicit method to construct such distributions with given first three moments using minimal number of phases. In the following, we focus on the class of minimal APH distributions (MinAPH). Note that APHM renewal processes would destroy the skip-free property of the queueing process when used as arrival or service process.

Depending on the second and third normalized moment, the aimed-at distribution is either an Erlang-Exp distribution or an Exp-Erlang distribution with parameters N, p, λ_X and λ_Y. They are shown in Figure 5.3 and Figure 5.4, respectively.

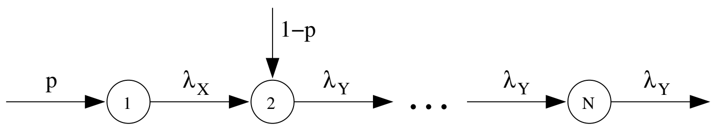


Figure 5.4: The underlying Markov chain of the Exp-Erlang distribution

Moment matching for $m > 3$

Based on the three-moment matching, Johnson *et al.* developed a more general fitting procedure based on Nonlinear-Programming (NLP) [57, 59]. In this approach, the fitting is specified as an NLP problem and the task is to find the optimal Erlang-mixture that minimizes the distance of the resulting PH distribution and the input values. The limitation to mixtures of two common-order Erlang distributions has been removed: now, any number of Erlang distributions of different order is allowed. The procedure has the following properties:

- The fitting is not limited to three moments. The software package presented in [57] is able to fit to six moments and to other properties, e.g., specific values of the distribution function.
- The PH distribution returned by the NLP search may be smaller than the one found by the three moment matching method.
- The NLP search consumes much more computational resources than simple two- or three-moment matching methods. Additionally, it may require user interaction for optimal results. Heuristics are used to speed up the search and to minimize the size of the result.

The last property implies that it is a sensible decision to limit traffic descriptors to three moments, at least in the context of our queueing network analysis where the fitting has to be done at each node in every iteration.

5.6 PH renewal processes as traffic descriptors?

In the previous section we fitted PH renewal processes to the m-moment traffic descriptors. In this section, we want to examine if it is possible — and useful — to avoid the PH-fitting step by using the PH processes themselves as traffic descriptor.

What are the potential benefits? As already stated, the fitting step is very critical for moments higher than the third. Using PH renewal processes as traffic descriptors themselves would allow us to represent the distribution function of the inter-arrival time with an accuracy that cannot be reached even by three-moment traffic descriptors. Unfortunately, PH-based traffic descriptors suffer from a problem that neutralizes their expected benefits: similarly to the MAP-based approach (see

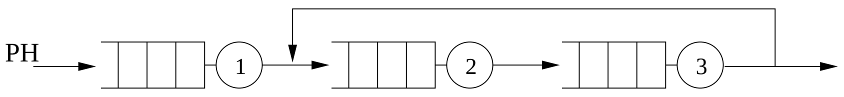


Figure 5.5: Usage of PH-based traffic descriptors

Chapter 9), the state-space explosion occurs during the superposition and service operation steps. The traffic descriptors would quickly become far too large since the superposition is repeated in each turn of the fixed-point iteration. To prevent the state space explosion we would be forced to fall back to approximations like the m-moment method.

Nevertheless, in some situations PH descriptors are still useful. Consider the network shown in Figure 5.5. We can specify the external arrival traffic to the first queue as a PH renewal process since no superposition occurs at this place and still analyze the other queueing stations by a moment-based method like FiFiQueues. Such an approach can be considered as a *hybrid* network analyzer that relies on two different kinds of traffic descriptors.

5.7 Summary and conclusions

In this chapter, we have discussed analysis methods that are based on so-called first-order traffic descriptors. We have started with a discussion of two well-known analysis method, namely the method of Jackson for networks of M|M|1 stations and Whitt's QNA. We have shown that the involved steps can be elegantly described in our decomposition framework. Both methods are restricted to queueing stations with infinite queueing capacity. This limitation has been removed by our analysis method called *FiFiQueues*. In addition, FiFiQueues replaces some of the approximations of QNA by exact solutions and allows arbitrary phase-type renewal processes as service processes.

However, unlike QNA, FiFiQueues requires a fixed-point iteration in order to compute the traffic flows of a queueing network. We have found that a complete analysis of the iteration behavior is difficult because the hypo-exponential PH distribution used by FiFiQueues has transition rates that are discontinuous as function of the squared coefficient of variation. We have presented a small modification to the original PH distribution that has allowed us to prove the existence of the fixed point. Further studies are required to decide whether the fixed point is unique and always reached. Concerning the original FiFiQueues algorithm, it is an interesting question how the discontinuity of the original PH distribution affects the existence of the fixed point. Although no proof is currently available, the properties of the original PH distribution (as illustrated in Figure 5.1) suggest that the distribution of the resulting QBD is discontinuous, too. Nevertheless, the original FiFiQueues always terminated in our experiments, so it seems that either the discontinuity is too

	JQN	QNA	FiFiQueues	m moments	PH-desc.
fixed-point iteration	no	no	yes	yes	yes
traffic descriptor	$\langle\lambda\rangle$	$\langle\lambda, c_A^2\rangle$	$\langle\lambda, c_A^2\rangle$	$\langle a_1, \ldots, a_m\rangle$	PH[1)]
finite q. capacity	no	no	yes	yes	yes
service process	μ	(μ, c_S^2)	PH	PH	PH
traffic merging	exact	approx.	approx.	approx.	exact
node analysis[2)]	exact	approx.	exact	exact	exact
traffic splitting[3)]	exact	renewal	renewal	renewal	renewal

[1)] only for external traffic descriptors

[2)] includes the analysis of the service operation and the node performance

[3)] "renewal" = exact if the traffic is generated by a renewal process

Table 5.1: Characteristics of the analysis methods with first-order traffic descriptors

small to affect the fixed point behavior of the algorithm in practice or the continuity is simply not required for the existence of a fixed point. This question will be a topic for further research.

Like QNA, FiFiQueues' traffic descriptors are based on the first and second moment of the inter-arrival time. We have also discussed the possibility to extend the traffic descriptors to higher moments. Especially for three moments, attractive PH-fitting algorithms are available. Finally, we have briefly shown how arbitrary PH renewal processes can be used as descriptors of external incoming traffic. We summarize the key features of the discussed analysis methods in Table 5.1. The performance of FiFiQueues will be discussed in the next chapter.

Chapter 6

Performance of FiFiQueues

In this chapter we evaluate the performance of the FiFiQueues algorithm with regard to the quality of the numerical results. This evaluation consists of

- tests with the original FiFiQueues algorithm (based on two-moment descriptors) on some representative queueing networks (Section 6.1),
- a case study of a web server (Section 6.2), and
- tests with three-moment traffic descriptors (Section 6.3).

We conclude this chapter with Section 6.4. Note that the third part, Section 6.3, also contains many performance results for the original FiFiQueues algorithm. However, the results are very detailed and some of the test models have been designed for the comparison with the three-moment traffic descriptors.

The results of the numerical analysis are compared to results determined using the discrete-event simulator described in Chapter 8.3. The relative half-width of the 95%-confidence intervals is smaller than 1% for all the simulation results. If not stated otherwise, arrival time and service time distributions specified by their rate and the squared coefficient of variation (SCV) are always mapped to the PH distributions using the original FiFiQueues algorithm. Relative errors between numerical analysis and simulation are always computed relative to the latter.

6.1 Evaluation of FiFiQueues

In this section we evaluate FiFQueues' performance with some typical networks. We begin with a single queue in Section 6.1.1, then continue with some complex networks in Section 6.1.2, and finally, in Section 6.1.3, we study the impact of the modified hypo-exponential distribution as introduced in Section 5.4.5. The presented tests cover a wide range of input parameters, including (nearly) deterministic processes, and complex networks with finite queueing capacities.

load	analysis	simulation	rel. error
0.1	0.10	0.10	0.0%
0.2	0.20	0.20	0.0%
0.4	0.47	0.47	4.4%
0.6	0.95	0.89	6.7%
0.8	2.34	2.18	7.3%
0.95	10.60	9.26	14.4%

Table 6.1: Mean queue length for a queueing station with deterministic arrival traffic

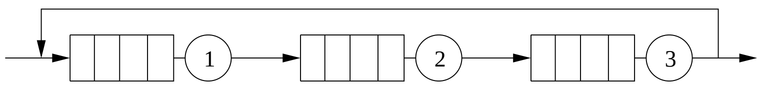


Figure 6.1: 3-node queueing network with feedback

6.1.1 Single queues

In the case of queueing networks that consist of only one queueing station FiFi-Queues always produces exact results, provided that the selected arrival and service PH renewal processes match the actual arrival and service processes of the real system. Hence, results of single-queue systems are not very interesting. At this place, we will only discuss the special case of deterministic distributions.

As explained, FiFiQueues limits the number of phases in hypo-exponential PH distributions to 10, which corresponds to a minimum SCV of 0.1. As a consequence, deterministic distributions can only be approximated. To evaluate the effect of this restriction we have analyzed a queueing station with negative exponential services and deterministic arrival process at different loads. Table 6.1 compares the so obtained mean queue lengths with results found by simulation. It shows that the relative error between analysis and simulation increases with the load. Errors of comparable magnitude can also be observed for other performance measures and for hypo-exponential and hyper-exponential service distributions. Note that, in general, FiFiQueues' pessimistic estimations are not particularly disturbing since we are often interested in the worst-case performance of a queueing network.

6.1.2 Queueing networks with feedback

3-node queueing network

We first address three queueing nodes in series, with a feedback from the last to the first queue, as shown in Figure 6.1. The external Poisson source has rate 1.3 and the service times are Erlang-5 distributed with rate 1.5; the node capacity is 10 (not including the service station) at all queues. The feedback probability is 25%.

The results are shown in Table 6.2. The first two rows show the characteristics

		analysis	simulation	rel. error
Output traffic	$\lambda_{netd,3}$	1.08	1.08	0.0%
	$c^2_{netd,3}$	0.41	0.41	0.0%
Arrival traffic	$\lambda_{a,1}$	1.65	1.66	-0.6%
	$c^2_{a,1}$	0.96	0.96	0.0%
	$\lambda_{a,2}$	1.47	1.47	0.0%
	$c^2_{a,2}$	0.23	0.23	0.0%
	$\lambda_{a,3}$	1.45	1.45	0.0%
	$c^2_{a,3}$	0.21	0.22	-4.5%
Queue length	E$[N_1]$	6.47	6.47	0.0%
	E$[N_2]$	4.43	4.45	-0.4%
	E$[N_3]$	3.96	3.90	1.5%

Table 6.2: Results for the 3-node network with Poisson source

		analysis	simulation	rel. error
Output traffic	$\lambda_{netd,3}$	0.99	0.99	0.0%
	$c^2_{netd,3}$	0.45	0.69	-34.8%
Arrival traffic	$\lambda_{a,1}$	1.63	1.63	0.0%
	$c^2_{a,1}$	3.33	2.35	41.7%
	$\lambda_{a,2}$	1.33	1.33	0.0%
	$c^2_{a,2}$	0.79	0.79	0.0%
	$\lambda_{a,3}$	1.32	1.33	-0.8%
	$c^2_{a,3}$	0.35	0.65	-46.2%
Queue length	E$[N_1]$	5.57	5.59	-0.4%
	E$[N_2]$	3.30	3.16	4.4%
	E$[N_3]$	2.38	2.76	-13.8%

Table 6.3: Results for the 3-node network with hyper-exponential source

of the traffic leaving the queueing network from node 3. The middle six rows show the rate and SCV of the arrival traffic at each node, and the last three rows show the expected queue length at each node.

The good results of the analysis can be explained by the fact that the resulting arrival traffic to node 1 (i.e., where the traffic superposition operation happens) is near to Poisson as indicated by $c^2_{a,1}$=0.96. If we replace the external source distribution by a hyper-exponential distribution with $c^2 = 4.0$ we obtain the results shown in Table 6.3. As expected, larger errors can be observed this time for the SCV of the arrival traffic. Interestingly, node 2 does not seem to be affected. This is because node 2 is fed by node 1 which is overloaded and hence reduces short-range correlations in the traffic stream.

Figure 6.2 shows the incoming traffic to node 1 as a function of the number of iterations in the fixed-point procedure for both kind of external sources. As can be

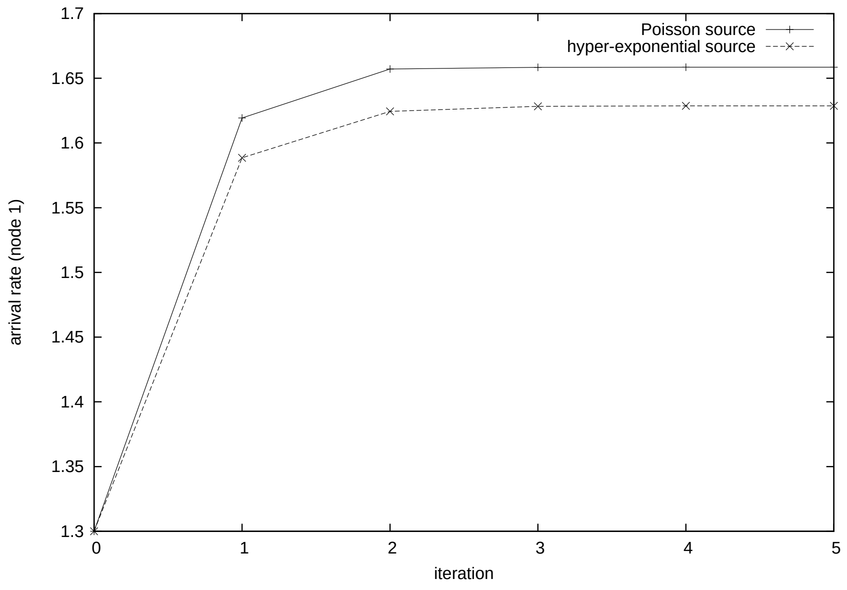


Figure 6.2: Incoming traffic to node 1 as a function of the number of iterations in the fixed-point procedure for the 3-node network

observed, the fixed-point is reached after a very small number of iterations. This behavior has been typical for all queueing networks we have analyzed so far.

Kühn's nine-node network

As a larger queueing network we evaluated a modified version of Kühn's nine-node network [64], as shown in Figure 6.3 (the numbers at the edges specify the routing probabilities). A similar network has been examined in [45, 113]. The external arrival rate to nodes 1–3 equals 0.8 and $c^2_{ext} = 4.0$. The service rate at each node is 1.0 (except for node 5 where $\mu_5 = 0.5$), and the SCV of all service processes is $c^2_s = 0.5$. All nodes have a finite queueing capacity of 25. Hence, without decomposition the underlying CTMC would comprise $2^3 \cdot (1 + 25 \cdot 2)^9 \approx 1.86 \times 10^{16}$ states.

Table 6.4 shows the results obtained by FiFiQueues and by simulation for the mean queue length and the offered load at each station. Note that the results for the (identical) nodes 1–3 are only stated once.

6.1.3 The modified hypo-exponential PH distribution

So far, we have done all tests with the original FiFiQueues version as described in Section 5.3. Now, we want to study the effect if we replace the hypo-exponential PH distribution of the original FiFiQueues by the new hypo-exponential PH distribution that we have introduced in Section 5.4.5. We reuse the nine-node network of the

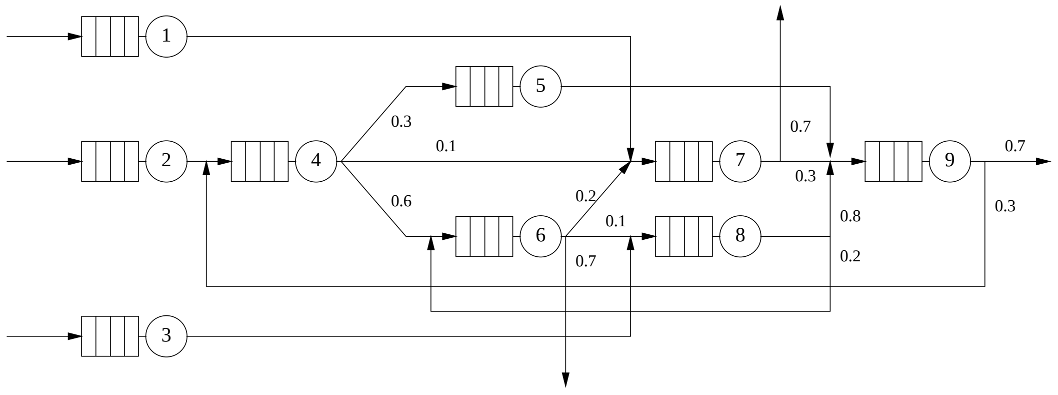


Figure 6.3: Kühn's nine-node network

node		analysis	simulation	rel. error
1–3	E[N_i]	6.39	6.39	0.0%
	offered load	0.8	0.8	0.0%
4	E[N_4]	16.84	16.74	0.6%
	offered load	1.09	1.09	0.0%
5	E[N_5]	1.14	1.13	0.9%
	offered load	0.59	0.59	0.0%
6	E[N_6]	2.31	2.28	1.3%
	offered load	0.77	0.76	1.3%
7	E[N_7]	14.67	14.86	-1.3%
	offered load	1.04	1.04	0.0%
8	E[N_8]	6.36	6.63	-4.0%
	offered load	0.87	0.87	0.0%
9	E[N_9]	22.41	21.88	2.4%
	offered load	1.28	1.27	0.8%

Table 6.4: Results for the departure rates in Kühn's nine-node network

previous section but we modify the SCV of the service processes to make the test more significant: the first node gets a c^2 of 0.1, the second node 0.2, the third node 0.3, and so on.

The analysis shows that the original FiFiQueues and the modified version compute exactly the same values for the departure rates (relative difference less than 10^{-7}). The largest relative difference of the SCV of the departure traffic is less than 10^{-4}, so we can conclude that the new hypo-exponential distribution does not affect the quality of FiFiQueues. We therefore do not show separate numbers in this thesis.

6.2 Performance evaluation of a web server

In this section we will use FiFiQueues for the performance evaluation of a web server. The employed parameters in the models have been derived from measurements made at a test system. This section is structured as follows. First, we describe the test system in Section 6.2.1. Then we present a QN model for a web server without disk access (cache only) in Section 6.2.2, followed of the model of a web server with disk access in Section 6.2.3. These two models are then combined to a model of a server group in Section 6.2.4. We compare the results obtained by analysis with the results obtained by simulation, and, where available, with the data collected at the test system.

6.2.1 Description of the test system

The test system consists of a computer running the Apache web server [4]. The server load is generated by two client systems that send HTTP/1.0 GET requests to the server in a 100 MBit Ethernet LAN (see Figure 6.4). The request times as well as the sizes of the requested files have been extracted from traces (access logs) collected at the UC Berkeley Home IP Service [41] in 1996. For our tests we have used a part of the original trace file: it consists of 35541 requests for static files (i.e., pictures, html-pages, etc.) sent over 4 hours by different users. This corresponds to a request rate of 2.468 requests per second. The SCV of the inter-request time is 1.2. The requested files have a mean size of 8510 bytes where the smallest file has a size of 2 bytes and the largest file a size of about 4.5 MBytes. The size distribution has a SCV as large as 26.8.

The web server of the test system has been configured to use not more than 150 server threads. This implies that the number of requests that can be processed concurrently is limited to 150. Since connection requests are not queued the clients will experience a connection rejection if they try to exceed this number. In addition, the request time out has been set to 8 seconds. More details concerning the test system can be found in [61]; please note that the QN models presented in the

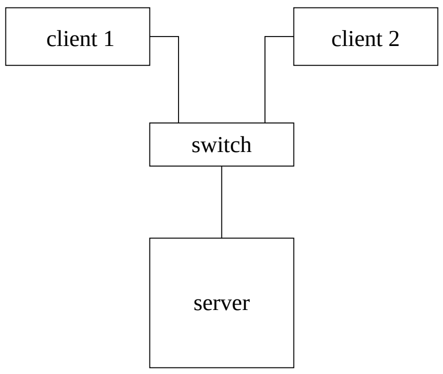


Figure 6.4: Topology of the test system

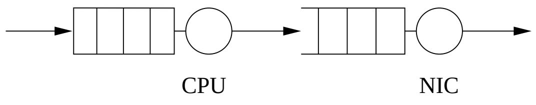


Figure 6.5: QN model for the web server without disk access

following differ from the models discussed there.

6.2.2 Web server without disk access

For the first model, we assume that the server holds all requested files in the file cache and, as consequence, no disk access is performed. This is a typical situation in intranets where the number of often requested files is limited. In this scenario the performance of the web server is only limited by the CPU, the main memory, and the network interface controller (NIC).

We model the web server by two queueing stations in series as shown in Figure 6.5. Both stations have a finite queueing capacity. The first station is fed by an external source that represents the clients sending the HTTP requests. The SCV of 1.2 for the source is equal to the corresponding value of the trace file.

The first station models the CPU. Measurements at the test system have shown that the CPU of the test server is able to process up to around 1200 requests per second. We adopt this value for the service rate of the first queueing station. Concerning the SCV of the CPU's service time distribution, we observe that the CPU service time is dominated by the time to handle the HTTP protocol and by the management of the cache data structures. Since the NIC accesses the main memory via DMA (Direct Memory Access), the CPU service time exhibits nearly no dependency on the size of the requested file. Hence, we choose a (nearly) deterministic service time distribution with a SCV of 0.1. The second queue represents the NIC. Measurements have shown a network load between 90% and 95% for a response rate of 1100 responses per second. This leads to a NIC service rate of approximately

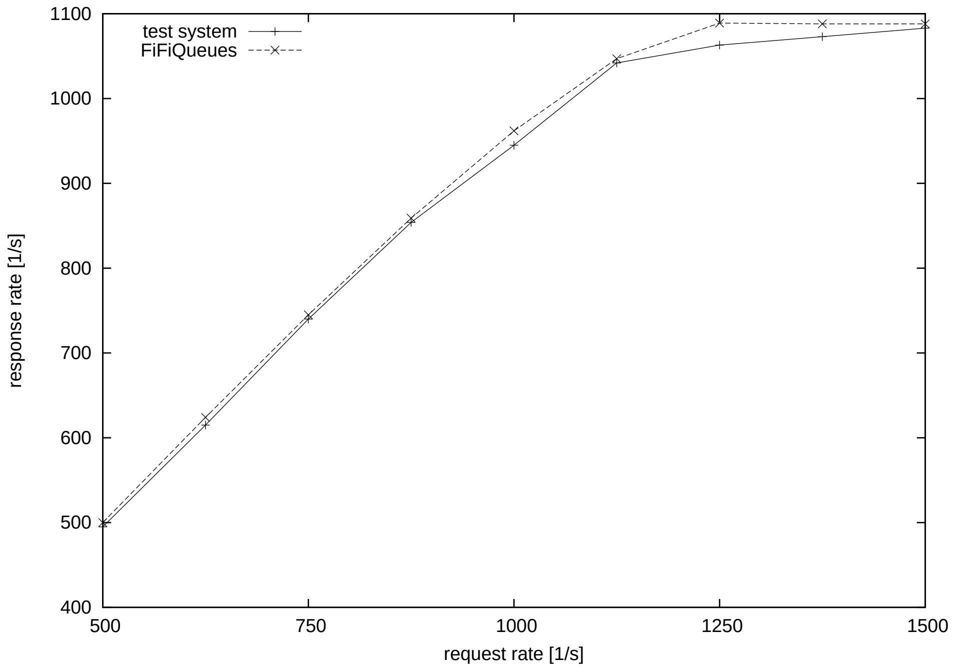


Figure 6.6: Response rate as function of the request rate for the web server without disk access

1200. For the SCV of the NIC's service time distribution, we assume a direct dependency of the service time on the file size and we set the SCV to 26.8, i.e., to the SCV of the file size distribution.

The most problematic aspect of the test system is the limitation to 150 simultaneously connected clients. This cannot be easily modeled by the FCFS-scheduling used by FiFiQueues. To approximate the limit, we have first analyzed the network at a request rate of 1500 requests per second. By a Newton-iteration, we have determined the queueing capacity at which the total number of jobs in the network is 150. The thus found capacity of 106 has then been used for all other request rates (we have chosen the same capacity for both queues; the jobs are distributed evenly over both stations at high request rates).

Figure 6.6 shows the number of responses per second as function of the number of requests sent per second as measured at the test system and as computed by FiFiQueues. Simulation results are not shown since they are nearly identical to the analytical results (relative error $< 1\%$). It shows that the QN model is able to predict the response rate quite well. The total mean response times are shown in Figure 6.7. The results are acceptable, but we can see that the model is not able to reproduce the sharp jump of the response time at 1000 requests/s. A model with more complex behavior, for example non-FCFS scheduling, would be required in order to obtain better results.

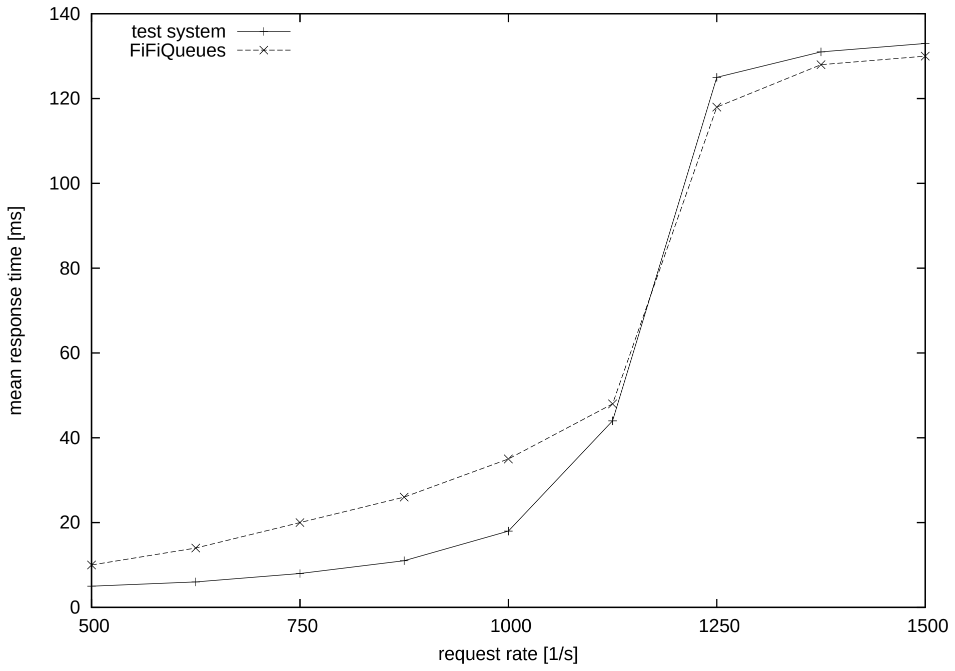


Figure 6.7: Mean response time as function of the request rate for the web server without disk access

6.2.3 Web server with disk access

The second model assumes that all requested files have to be loaded from the disk of the server system. Measurements have shown that the test system only achieves a maximum response rate of 63.5 files/s at a CPU load of 9%. Clearly, the disk transfer is the bottleneck.

We model the influence of the disk access through an additional queueing station. Figure 6.8 shows the resulting model. The first station represents the CPU. For the SCV of the CPU service time, we have kept the value of 0.1 of the previous model. However, the service rate has now been set to a value of 706 ($= \frac{63.5}{0.09}$) to reflect the higher CPU demand of the single disk-based request. The service rate of the disk station has been set 63.5. For the SCV, we have assumed a direct dependency of the service time on the size of the requested file *measured in blocks of 4 KBytes* since this corresponds to the organization of the data on the disk. This leads to a SCV of 16.5 instead of 26.8. The NIC in this model has the same service rate and SCV as in the previous model.

Again, the problem of the bounded number of simultaneously connected clients remains. Since the disk station clearly is the bottleneck, we have limited its queueing capacity to 150 while the CPU and the NIC station now have infinite queueing capacity. Note that, in spite of the large differences between the service rates, the CPU and the NIC should not be removed from the model since they have a small

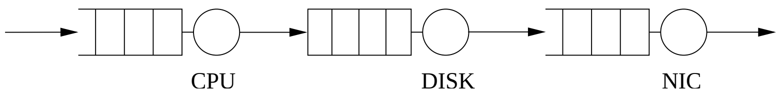


Figure 6.8: QN model for the web server with disk access

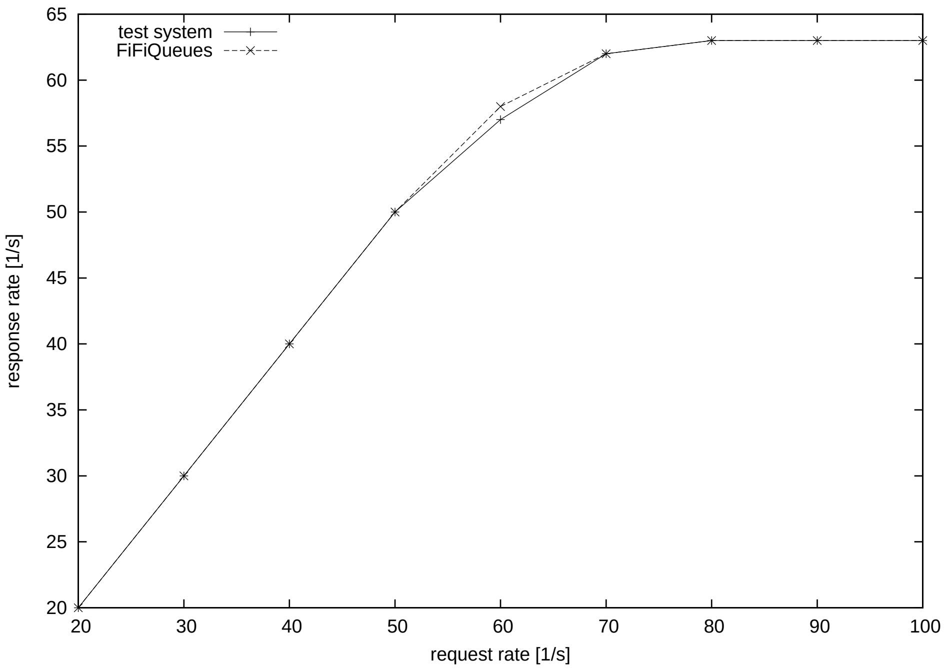


Figure 6.9: Response rate as function of the request rate for the web server with disk access

but measurable influence on the SCV of the traffic stream.

Figure 6.9 shows the number of responses per second as function of the number of requests sent per second as measured at the test system and as computed by FiFiQueues. Again, simulation results are not shown since their are nearly identical to the analytical results (relative error $< 1\%$). Again, it shows that the QN model is able to predict the response rate quite well. The total mean response times are shown in Figure 6.10. We observe that the QN model underestimates the response time, especially at request rates near to the maximum response rate of the disk. Our experiments with more complex QN models have shown that an improvement of the results cannot be easily achieved by using the type of queueing stations offered by FiFiQueues. For example, a more appropriate model would have to consider that the seek time of the disk becomes a significant part of the disk's response time at high file reqest rates since the disk has to reposition its read/write heads more often. Detailed models like the one presented in [98] simulate axial and rotational head positions, seek, rotation and transfer times, and provide separate submodels for the disk mechanism, the cache and the DMA engine.

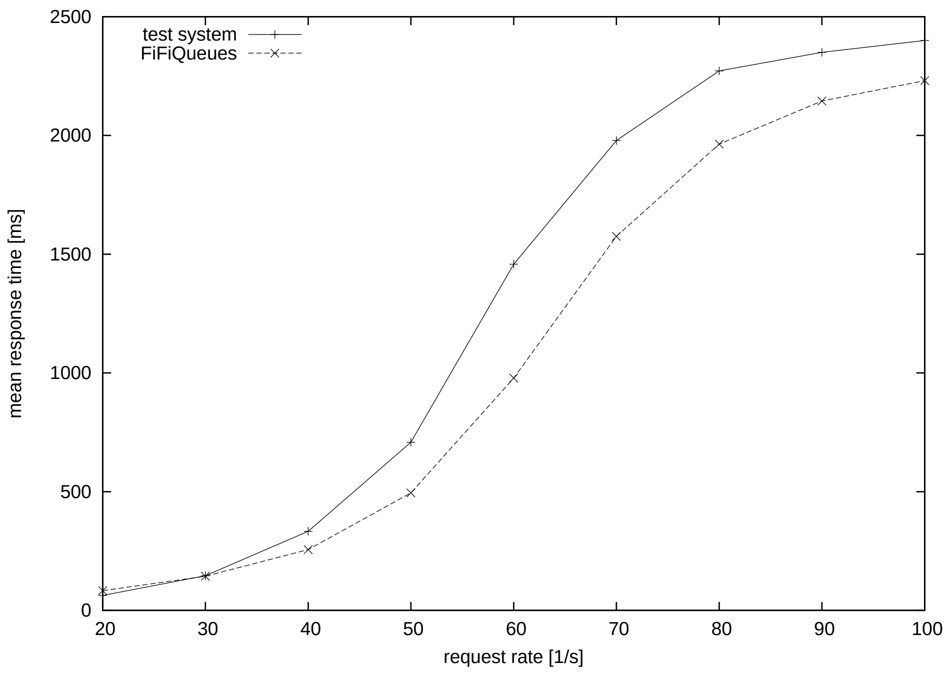


Figure 6.10: Mean response time as function of the request rate for the web server with disk access

6.2.4 Group of servers

In this section we evaluate a group of servers as shown in Figure 6.11. In our model, the client requests HTML pages from the main server of a web site. An HTML file refers to, in average, three other objects (company logo, images,...) that are also located on the main server. In addition, the HTML file refers to an object located on one of the five data servers. We assume that the HTML file and the three referred files located on the main server are frequently requested and, hence, the main server mainly operates on the cache. Concerning the data servers, we assume that they store large amounts of infrequently requested files, for example files specific to the requesting user, media files, et cetera. The client uses the HTTP/1.0 protocol [83], i.e., the five files that constitute the requested HTML page are sequentually requested.

The QN model is shown in Figure 6.12. The QN of the server without disk access (representing the main server) is combined with five copies of the QN of the server with disc access (representing the data servers). Jobs leaving the main server are fed back to it with a probability of 0.75, thus resulting in four visits to the main server in average. The jobs finally leaving the main server are distributed evenly on the data servers. The service processes and the capacities of the stations remain unchanged.

We have evaluated the QN model by FiFiQueues and by simulation. The results

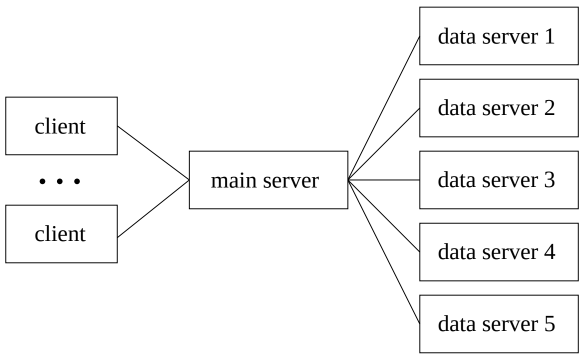


Figure 6.11: Group of Web servers

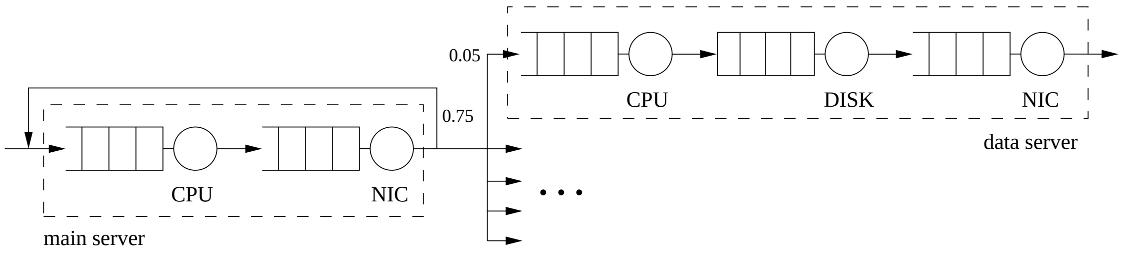


Figure 6.12: QN model for the server group

for the response rate (for one data server) and the mean response time are shown in Figure 6.13 and, respectively, Figure 6.14. The vertical bars in the latter show the 95%-confidence intervals of the simulation results. FiFiQueues provides good results for request rates smaller than 250. At larger request rates, FiFiQueues overestimates the losses in the main server because it ignores the correlations caused by the feedback. As consequence, the load of the data servers is underestimated which leads to a smaller mean response time in comparison with the results obtained by simulation.

Table 6.5 shows the runtimes (in seconds) of the FiFiQueues algorithm and of the discrete-event simulation for the evaluation of the server group model with various request rates. For FiFiQueues, we have recorded the runtimes for two different implementations of the finite queue analysis. The original implementation uses a Gauss-Seidel iteration, whereas a new version uses the Cyclic Reduction method

	FiFiQueues		
request rate	Gauss-Seidel	Cyclic Red.	simulation
100	7	2	11
200	15	3	11
300	19	3	11

Table 6.5: Runtimes (in seconds) for the evaluation of the server group model

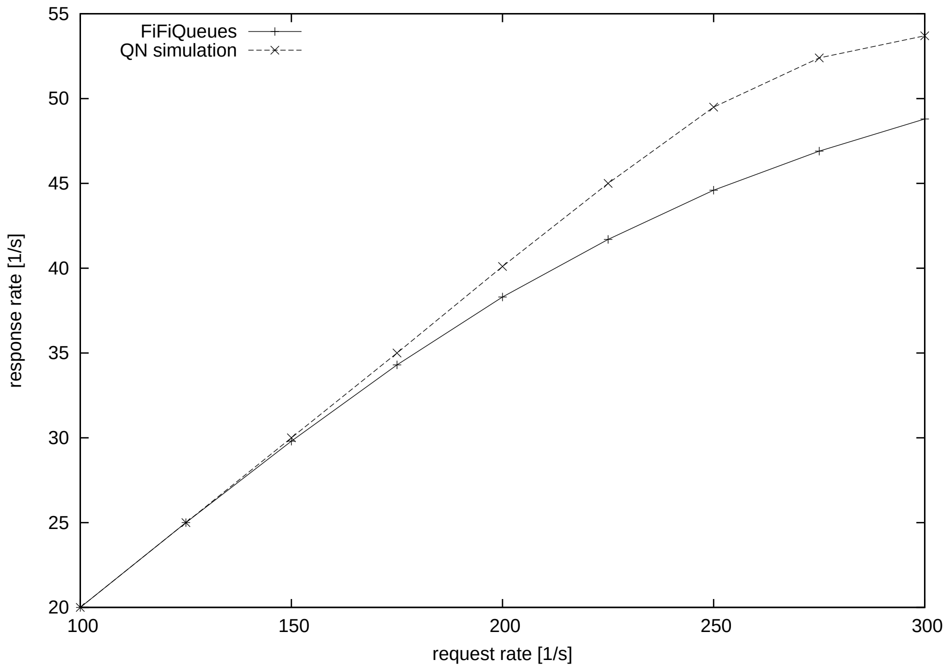


Figure 6.13: Response rate as function of the request rate for the server group

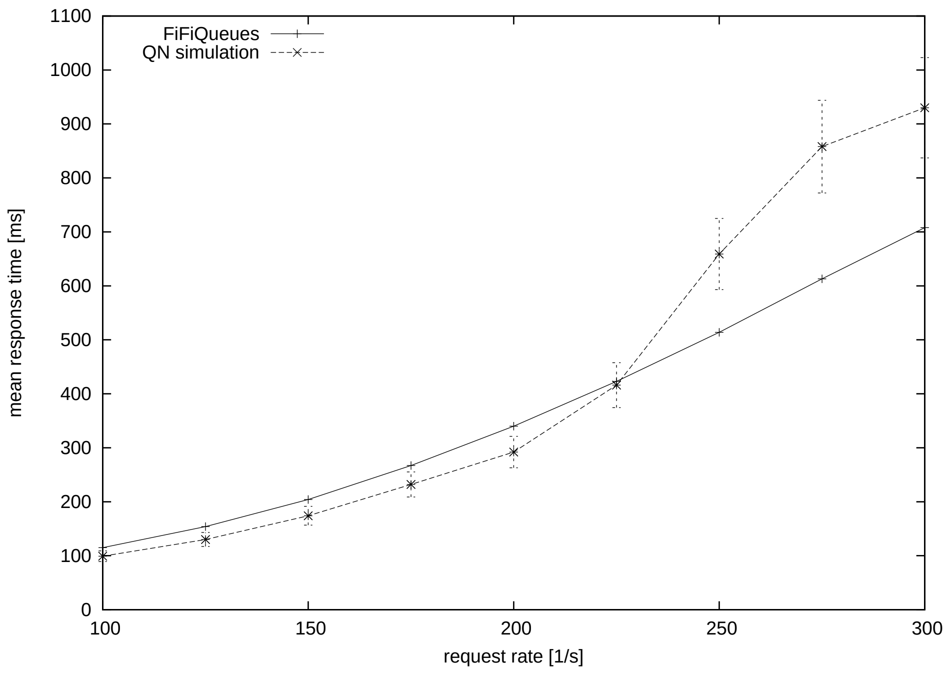


Figure 6.14: Mean response time as function of the request rate for the server group

(see Section 3.4.3). As mentioned in Section 3.5, the runtime of the Gauss-Seidel iteration increases with the load of the stations. The Cyclic Reduction method is clearly faster than the Gauss-Seidel iteration and the simulation.

6.3 FiFiQueues with three-moment descriptors

In this section we analyze the effect of the third moment as part of the traffic descriptor. Basically, two types of models can be identified:

1. Models that only specify the first and second moment of the external arrival processes, as supported by FiFiQueues.
2. Models also containing information about the third moment. These models cannot be analyzed by the original FiFiQueues algorithm.

In the following, our focus is on models of the first type because we are interested in the question whether three-moment descriptors yield better results than the original FiFiQueues algorithm *for the same model*. The benefit of three moments in the model specification, as addressed by the models of the second type, has been already intensively discussed in former publications (see [58] for an overview).

When fitting a PH distribution to three moments, the three methods presented in Section 5.5.5 are available. In the following we denote the fitted PH distributions by PH_{Em} (Erlang-mixture), PH_{EC} (Erlang-Coxian), resp. PH_{Min} (Minimal Acyclic PH, MinAPH). We start with a comparison of these PH distributions in Section 6.3.1. In Section 6.3.2 we discuss some simple tandem queueing networks. The traffic splitting operation is studied in Section 6.3.3. Finally, we evaluate the traffic merging operation in queueing networks with feedback in Section 6.3.4. Some auxiliary test results can also be found in Section 10.3.

6.3.1 Comparison of the distributions

For the comparison, we have used the three methods to fit PH distributions to different SCVs in the range $[0.1, 4]$ and to skewnesses in the range $[-3, 6]$. The SCV and the skewness are denoted as c^2, respectively γ, in the following. The Erlang-mixture method allows to choose the common order n of the mixed Erlang distributions. We have chosen $n = n^*$, as defined in Section 5.5.5, for the first test.

In Figure 6.15, we show the ratio $\frac{size(PH_{Em})}{size(PH_{Min})}$ where $size(\cdot)$ gives the number of phases of a PH distribution. The part of the plot where this ratio equals 0 represents the area $\gamma \leq c - 1/c$ that cannot be realized by a PH distribution. It shows that the PH_{Em} distributions are about twice as large as the PH_{Min} distributions for moderate γ and especially for hypo-exponential distributions. With regard to the absolute values of the sizes, Figure 6.16 shows the size of PH_{Em}, PH_{Min} clearly is

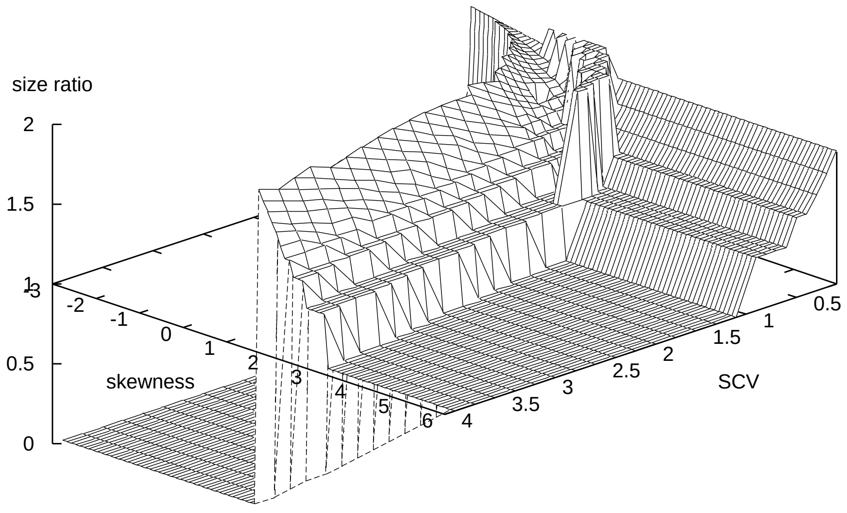


Figure 6.15: Ratio between the number of phases of the Erlang-mixture and of the Minimal APH distribution

the better choice. The size of the PH_{EC} distribution is not shown here since it is either equal to or by one larger than the size of the PH_{Min} distribution.

Another important aspect is the shape of the probability distribution function of the generated PH distributions. In general, distribution functions with extreme shapes or high sensitivity to the input parameters are not desired. To evaluate this we have computed the *kurtosis* (more correctly, the *kurtosis excess*) κ defined as the fourth standardized moment diminished by 3:

$$\kappa = s_4 - 3 = \bar{a}_4/\bar{a}_2^4 - 3.$$

The normal distribution has a kurtosis of 0. A high kurtosis indicates that the variance of the distribution is caused by infrequent, but extreme deviations. Figure 6.17, 6.18, and 6.19 show the kurtosis for the PH_{Em}, PH_{EC}, resp. PH_{Min} distribution. A value of –100 indicates the parameter range where $\gamma \leq c - 1/c$. We can observe that the kurtosis of PH_{Em} and PH_{Min} has more and sharper peaks whereas the kurtosis of PH_{EC} looks smoother. For PH_{Em}, this problem has been recognized in [58] and is caused by our decision to choose the smallest possible common order $n = n^*$ for PH_{Em}. If we choose $n = n^* + 1$ as common order, a kurtosis function with smaller peaks is obtained, as shown in Figure 6.20 and observed in [58]. In our experiments, larger values for n, e.g., $n = n^* + 2$, did not have a significant effect on the results. Hence, we will use $n = n^* + 1$ in the following.

But the most important observation is that the kurtosis of PH_{EC} is considerably higher than of PH_{Em} and PH_{Min}. This has also been observed in [12]. We will study in the following sections how this difference affects the obtained results.

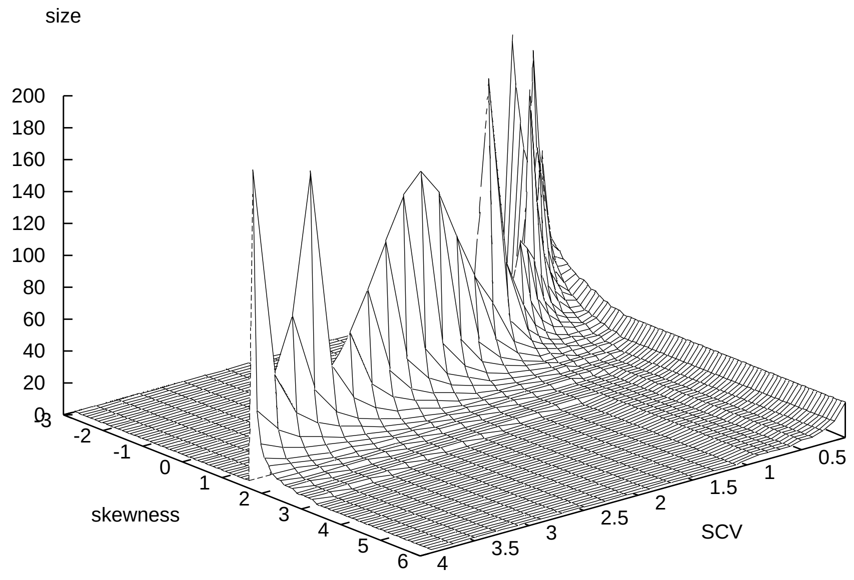


Figure 6.16: Number of phases of the Erlang-mixture distribution

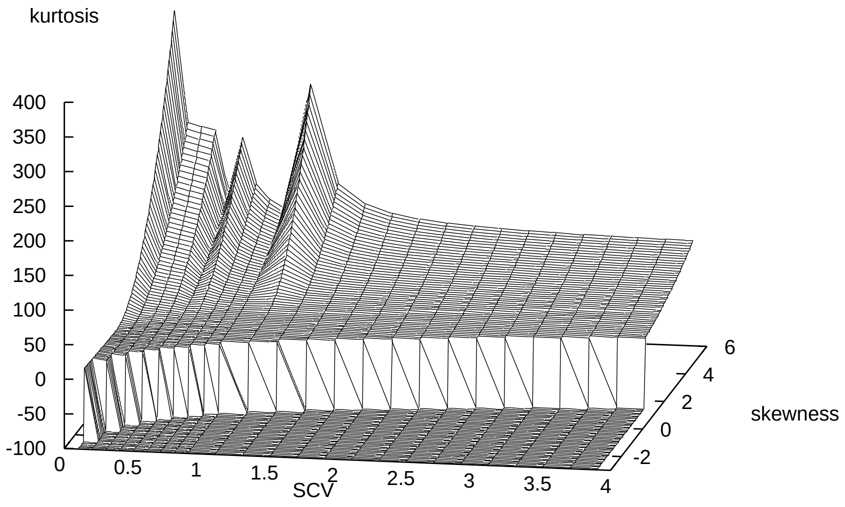


Figure 6.17: Kurtosis of the Erlang-mixture distribution with common order n^*

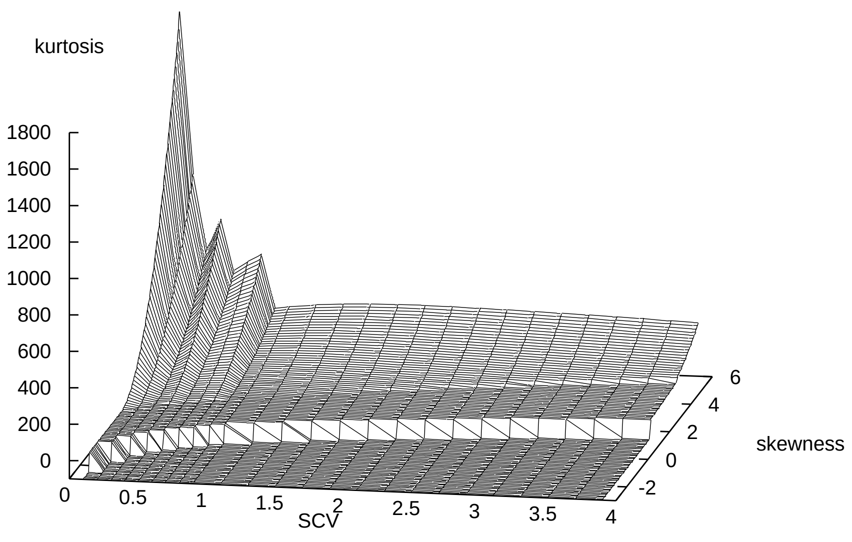


Figure 6.18: Kurtosis of the Erlang-Coxian distribution

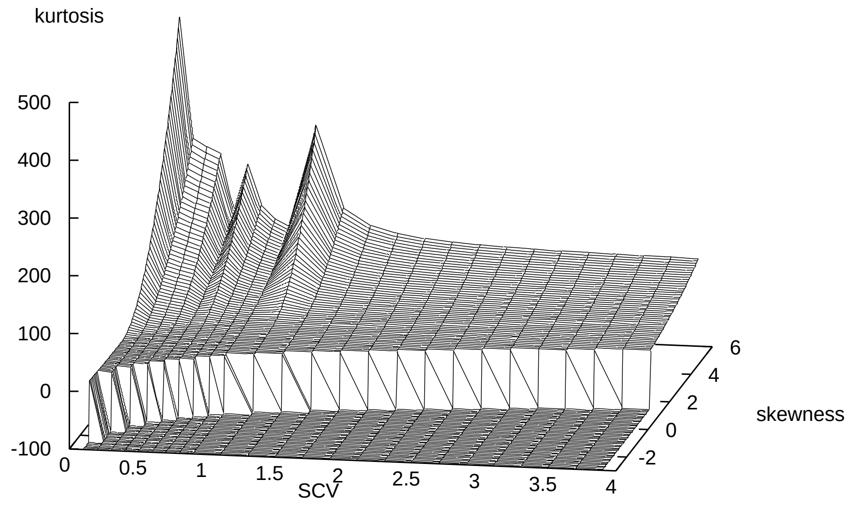


Figure 6.19: Kurtosis of the Minimal APH distribution

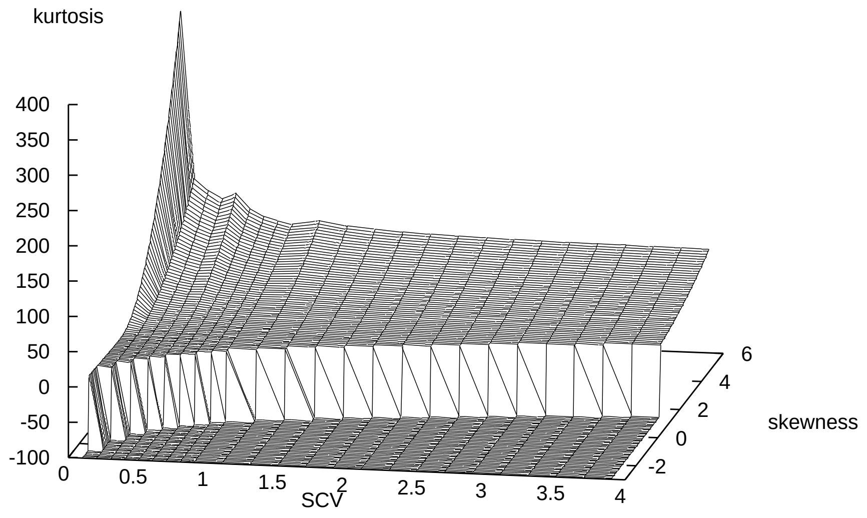


Figure 6.20: Kurtosis of the Erlang-mixture distribution with common order $n^* + 1$

6.3.2 Tandem queueing networks

The first model consists of three queues in series. All stations have a finite capacity of 10. The service rates are 1.5, 1.3, resp. 1.3, and the SCV of the service time distributions are 0.2, 0.2, resp. 2.0, for the first, second, resp. third station. The external arrival traffic to the first queue is specified by its rate λ_{ext} and its SCV c^2_{ext}. Table 6.6 shows the error (relative to simulation) of the mean queue lengths of the second and third queue as obtained by numerical analysis for various values of λ_{ext} and c^2_{ext} (with common order $n^* + 1$ for this and all following experiments). Note that the analysis of the first queue is always exact by design and hence its results are not shown in the table. The last four rows give the absolute values of the relative errors averaged over all values of λ_{ext} and both stations.

All methods yield acceptable, and similar, results for $c^2_{ext} \leq 1.0$ or for $\lambda_{ext} = 2.0$. In the other cases, PH_{Min} and PH_{Em} generally give the best results, followed by FiFiQueues. The PH_{EC} distribution generates the largest errors which is caused by the extreme kurtosis of its distribution function, as discussed in Section 6.3.1.

6.3.3 Traffic splitting

We study the traffic splitting operation by the model shown in Figure 6.21. The external arrival traffic to the first queue is specified by its rate λ_{ext} and its SCV c^2_{ext}. The SCV of the service time distribution of the first queue is denoted by $c^2_{s,1}$ and p

		FiFiQueues		Em		EC		MinAPH	
λ_{ext}	c^2_{ext}	$re(N_2)$	$re(N_3)$	$re(N_2)$	$re(N_3)$	$re(N_2)$	$re(N_3)$	$re(N_2)$	$re(N_3)$
0.5	0.5	0.7%	-0.3%	-0.6%	0.0%	4.5%	0.0%	2.5%	-0.5%
	1.0	4.8%	0.5%	-3.7%	-1.0%	6.7%	-0.5%	0.7%	-0.9%
	2.0	14.0%	5.9%	-3.4%	-1.6%	51.3%	9.5%	11.1%	2.6%
	4.0	18.9%	8.8%	-7.5%	-3.0%	104.0%	21.2%	14.4%	3.3%
1.0	0.5	-1.3%	-1.4%	-3.3%	-1.4%	0.7%	-1.9%	-2.7%	-1.4%
	1.0	-3.0%	-2.4%	-9.1%	-2.0%	-1.6%	-2.6%	-6.8%	-2.3%
	2.0	5.5%	-4.0%	-8.1%	-2.5%	4.6%	-4.1%	0.4%	-4.4%
	4.0	6.7%	-3.7%	-7.5%	-0.9%	6.0%	-11.5%	-2.1%	-2.1%
2.0	0.5	0.0%	0.1%	0.0%	0.1%	0.0%	0.1%	0.0%	0.1%
	1.0	-0.1%	0.3%	-0.1%	0.3%	-0.0%	0.3%	-0.1%	0.3%
	2.0	-0.3%	0.4%	0.1%	0.2%	0.2%	0.3%	0.1%	0.3%
	4.0	-7.6%	0.5%	-1.1%	0.8%	-7.2%	0.9%	-5.0%	0.9%
average	0.5	0.6%		0.9%		1.2%		1.2%	
relative	1.0	1.9%		2.7%		2.0%		1.9%	
errors	2.0	5.0%		2.7%		11.7%		3.2%	
	4.0	7.7%		3.5%		25.1%		4.6%	

Table 6.6: Mean queue lengths of three queues in series for different analysis methods

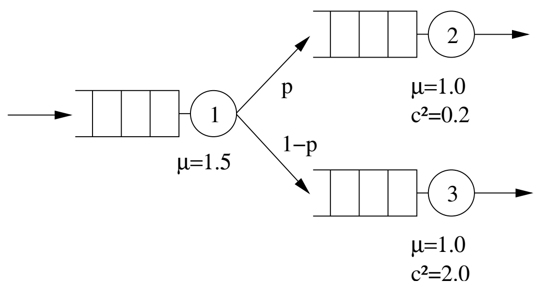


Figure 6.21: Queueing network to study the traffic splitting operation

gives the splitting probability for the departure traffic. All queueing stations have a capacity of 10. The other parameters are as shown in the figure (where μ and c^2 stand for the service rate, resp. service SCV). PH distributions are fitted to the rate and the SCV of the external source and the service time distributions by using FiFiQueues.

Table 6.7 and 6.8 show the errors relative to simulation of the mean queue lengths as computed by the different methods for $c^2_{s,1} = 0.5$, resp. $c^2_{s,1} = 2.0$, and for various values of p, λ_{ext}, and c^2_{ext}. The last three rows give the absolute values of the relative errors averaged over all values of p, λ_{ext}, and both stations. The results of FiFiQueues are satisfying. Only the PH_{Min} distribution gives a slight improvement. The larger errors obtained by the other methods can be explained by observing the standardized moments of the arrival time distribution at the queues. As an example, consider the situation where $p = 0.75$, $\lambda_{ext} = 0.5$, $c^2_{ext} = 4.0$, $c^2_{s,1} = 0.5$ from Table 6.7.

We find:

	c^2	skewness	kurtosis	$re(N_2)$
simulation	3.10	5.08	36.8	–
FiFiQueues	3.08	5.08	37.7	5.2%
Em	3.08	5.10	34.1	-13.0%
EC	3.08	5.10	126.1	73.6%
MinAPH	3.08	5.10	38.1	5.0%

Although small differences in the SCV and the skewness can be observed, the results indicate that the error is rather caused by the differences in the kurtosis of the employed PH distributions.

6.3.4 Traffic merging in queueing networks with feedback

We return to the queueing network with feedback from Section 6.1.2 in order to test the traffic merging operation. Again, we evaluate the network for a Poisson source and a hyper-exponentially distributed source. Table 6.9 shows the errors of the mean queue length relative to the simulation if we compute the first and the second moment of the merged traffic descriptor by Whitt's hybrid method and the skewness by the stationary-interval method. The results of FiFiQueues are given for comparison. The errors obtained by the three-moment descriptors are acceptable but we observe that they are generally larger than with FiFiQueues.

As a second example, we evaluate the server group model presented in Section 6.2.4 with three-moment descriptors. We use the MinAPH fitting method because it was the most promising in the previous experiments. Figure 6.22 shows the mean response time as function of the request rate as obtained by simulation, FiFiQueues, and three-moment descriptors. No improvement by the three-moment descriptors over FiFiQueues' two-moment descriptors can be observed.

In the two examples, three-moment descriptors have not shown a clear advantage in the analysis of queueing networks with feedback. The errors are mainly caused by the correlations in the traffic streams and, hence, can not be reduced by using three-moment descriptors. Additionally, it should be noted that the three-moment fitting does significantly increase the runtime of the analysis due to the larger PH distributions. For example, the hyper-exponential PH distribution used by FiFiQueues has two phases. In contrast, the arrival distributions generated by the MinAPH method for the first queueing station in the server group model have seven or eight phases, thus increasing the runtime of the analysis by a factor of 10 (Gauss-Seidel), respectively 6 (Cyclic Reduction) in comparison to the times measured for FiFiQueues (see Table 6.5).

			FiFiQueues		Em		EC		MinAPH	
p	λ_{ext}	c^2_{ext}	$re(N_2)$	$re(N_3)$	$re(N_2)$	$re(N_3)$	$re(N_2)$	$re(N_3)$	$re(N_2)$	$re(N_3)$
0.25	0.5	0.5	0.4%	0.1%	-2.4%	-0.4%	6.4%	4.8%	0.8%	0.6%
		1.0	3.7%	-0.5%	-2.1%	-3.2%	12.3%	0.7%	3.0%	-1.4%
		4.0	-0.9%	1.3%	-6.3%	-6.1%	42.1%	32.3%	1.2%	1.2%
	1.0	0.5	-1.8%	0.9%	-6.9%	0.9%	2.2%	1.4%	-1.6%	1.0%
		1.0	0.3%	-0.6%	-6.5%	-0.9%	5.2%	-0.7%	-0.4%	-0.8%
		4.0	2.9%	0.4%	-5.0%	-1.2%	27.6%	0.7%	3.8%	-0.8%
	2.0	0.5	-0.8%	0.1%	-5.9%	0.1%	1.6%	-0.1%	-0.8%	0.1%
		1.0	0.7%	0.2%	-4.5%	0.2%	3.1%	0.1%	0.7%	0.2%
		4.0	3.8%	-1.1%	-6.0%	0.2%	6.7%	-0.7%	-0.9%	-0.1%
0.50	0.5	0.5	-1.8%	-1.1%	-4.8%	-2.6%	-1.0%	-0.6%	-1.0%	-0.7%
		1.0	1.9%	0.1%	-6.0%	-3.8%	7.0%	2.8%	0.6%	-0.6%
		4.0	2.5%	-0.4%	-8.9%	-6.2%	68.4%	39.0%	4.5%	1.8%
	1.0	0.5	0.5%	-0.1%	-2.2%	-0.7%	1.1%	0.1%	0.9%	0.0%
		1.0	0.7%	-1.4%	-4.3%	-2.7%	1.7%	-0.9%	-0.6%	-1.8%
		4.0	8.6%	1.7%	-7.7%	-3.5%	35.6%	11.3%	5.4%	0.2%
	2.0	0.5	-0.9%	0.1%	-1.4%	0.0%	2.6%	0.7%	-0.9%	0.1%
		1.0	1.2%	-0.2%	0.7%	-0.3%	1.2%	-0.2%	1.2%	-0.2%
		4.0	5.5%	1.2%	-4.5%	-0.4%	2.9%	0.8%	-2.1%	-0.1%
0.75	0.5	0.5	-0.0%	-2.8%	-1.8%	-5.7%	10.6%	1.0%	1.2%	-2.6%
		1.0	1.6%	0.6%	-6.1%	-4.4%	4.4%	6.0%	-0.3%	0.2%
		4.0	5.2%	-0.9%	-13.0%	-4.5%	73.6%	31.7%	5.0%	2.1%
	1.0	0.5	0.9%	-0.3%	0.9%	-2.9%	3.9%	1.8%	1.4%	-0.2%
		1.0	-2.4%	-1.3%	-4.2%	-4.6%	-3.2%	1.2%	-3.6%	-1.6%
		4.0	4.5%	-0.9%	-5.2%	-4.7%	7.4%	13.7%	-1.0%	0.0%
	2.0	0.5	0.2%	0.6%	0.1%	-1.1%	0.5%	1.7%	0.2%	0.6%
		1.0	-0.1%	-0.2%	-0.2%	-1.9%	0.1%	0.9%	-0.1%	-0.2%
		4.0	-3.4%	1.3%	0.4%	-2.8%	-1.9%	2.4%	0.0%	-0.9%
average		0.5	0.7%		2.3%		2.3%		0.8%	
relative		1.0	1.0%		3.1%		2.9%		0.9%	
errors		4.0	2.6%		4.8%		22.2%		1.7%	

Table 6.7: Relative errors of the mean queue lengths for the traffic-splitting network with $c^2_{s,1} = 0.5$

			FiFiQueues		Em		EC		MinAPH	
p	λ_{ext}	c^2_{ext}	$re(N_2)$	$re(N_3)$	$re(N_2)$	$re(N_3)$	$re(N_2)$	$re(N_3)$	$re(N_2)$	$re(N_3)$
0.25	0.5	0.5	1.5%	-0.5%	-2.3%	-0.7%	9.1%	1.1%	2.5%	1.0%
		1.0	-3.0%	-1.7%	-9.0%	-3.1%	-1.6%	0.3%	-2.8%	-0.6%
		4.0	-1.1%	-3.2%	-6.3%	-8.6%	42.5%	26.8%	1.4%	-1.8%
	1.0	0.5	-0.9%	0.1%	-5.6%	-0.0%	6.7%	0.4%	-0.2%	0.1%
		1.0	-1.5%	-0.5%	-5.5%	-0.4%	1.4%	0.1%	0.7%	-0.2%
		4.0	-3.9%	0.1%	-9.1%	-0.1%	24.7%	0.9%	-0.2%	0.2%
	2.0	0.5	-3.0%	-0.4%	-5.1%	-0.5%	9.0%	-1.8%	0.6%	-0.7%
		1.0	-3.0%	-0.7%	-5.1%	-0.8%	0.6%	-2.0%	0.6%	-1.0%
		4.0	-3.6%	0.1%	-5.9%	0.0%	0.7%	-1.7%	0.7%	-0.3%
0.50	0.5	0.5	-2.8%	-1.2%	-6.8%	-3.0%	0.9%	0.9%	-0.7%	-0.0%
		1.0	-2.8%	-3.2%	-10.6%	-6.4%	-0.1%	-1.5%	-2.0%	-2.5%
		4.0	-6.5%	-5.0%	-16.1%	-9.3%	54.9%	33.4%	-3.5%	-1.7%
	1.0	0.5	-0.7%	1.6%	-3.5%	0.9%	4.1%	3.6%	0.0%	1.9%
		1.0	-3.6%	-3.3%	-3.7%	-3.0%	4.5%	-0.0%	-0.2%	-1.9%
		4.0	-6.2%	-2.1%	-12.0%	-3.1%	23.6%	10.3%	-2.2%	-0.2%
	2.0	0.5	-1.1%	-0.7%	0.6%	-0.5%	2.2%	0.0%	0.9%	-0.4%
		1.0	-0.7%	-0.0%	1.0%	0.1%	3.0%	0.8%	1.4%	0.3%
		4.0	-1.6%	-0.6%	0.7%	-0.2%	6.0%	1.4%	2.0%	0.2%
0.75	0.5	0.5	-4.2%	-1.5%	-5.9%	-5.1%	-0.9%	3.3%	-0.9%	-0.9%
		1.0	-4.4%	1.4%	-9.8%	-3.8%	-0.7%	2.5%	-2.7%	1.7%
		4.0	-7.4%	-7.0%	-21.5%	-9.8%	53.9%	24.3%	-5.3%	-3.5%
	1.0	0.5	3.7%	0.4%	3.1%	-1.6%	4.7%	4.8%	3.8%	1.0%
		1.0	-0.7%	-2.8%	1.1%	-4.1%	2.1%	-0.8%	1.3%	-1.2%
		4.0	-0.8%	-4.6%	-1.6%	-5.6%	3.5%	13.4%	-0.4%	-1.0%
	2.0	0.5	-0.8%	-2.4%	-1.2%	-2.3%	-2.9%	3.7%	-1.3%	-0.2%
		1.0	-1.1%	-0.5%	-1.6%	-0.4%	-3.4%	1.8%	-1.7%	1.7%
		4.0	0.5%	-2.6%	0.1%	-2.4%	-3.5%	0.2%	-0.3%	0.2%
average		0.5	1.5%		2.7%		3.3%		1.0%	
relative		1.0	1.9%		3.9%		1.5%		1.4%	
errors		4.0	3.2%		6.2%		18.1%		1.4%	

Table 6.8: Relative errors of the mean queue lengths for the traffic-splitting network with $c^2_{s,1} = 2.0$

c^2_{ext}	method	$re(N_1)$	$re(N_2)$	$re(N_3)$
1.0	FiFiQueues	0.0%	-0.4%	1.5%
	Em	-2.3%	0.4%	3.1%
	EC	-2.2%	0.2%	2.8%
	MinAPH	-2.3%	0.4%	3.0%
4.0	FiFiQueues	-0.4%	4.4%	-13.8%
	Em	-3.2%	-8.9%	-12.7%
	EC	-9.5%	-3.8%	-15.6%
	MinAPH	-3.8%	-6.0%	-12.2%

Table 6.9: Results for the queueing network with feedback and three-moment descriptors

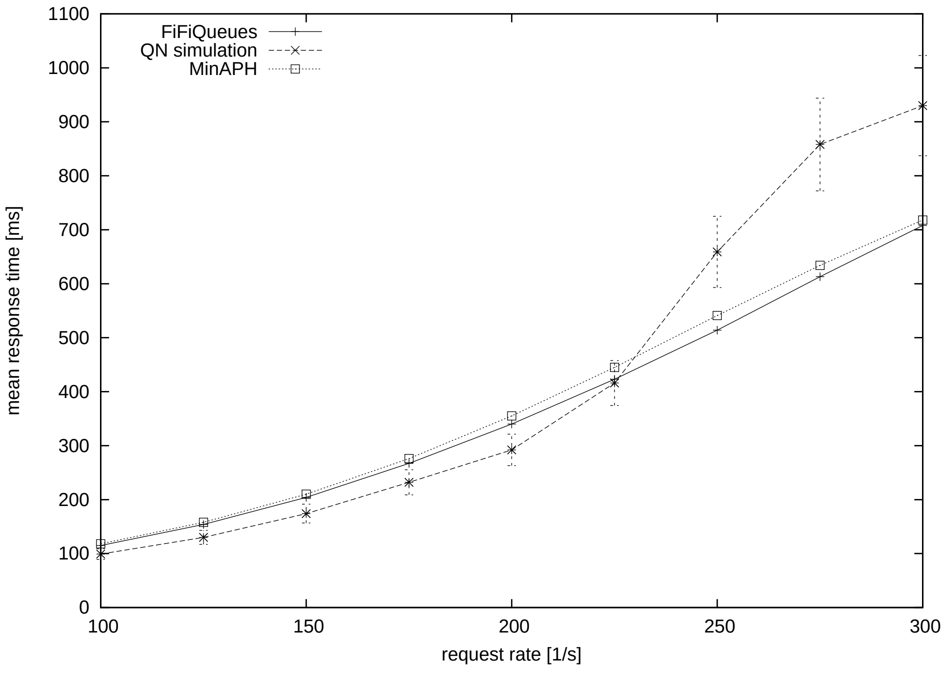


Figure 6.22: Mean response time as function of the request rate for the server group

6.4 Summary and conclusions

In this chapter, we have evaluated the performance of the FiFiQueues algorithm and its extension to three-moment traffic descriptors. Our experiments have shown that FiFiQueues provides very good results for important performance measures, like mean queue length, if the involved arrival times in the queueing network are hypo-exponentially or nearly (negative-)exponentially distributed. In such situations, we can generally expect relative errors less than 5%, even if the network has a complex structure. In case of hyper-exponential arrival processes, especially in queueing networks with feedback, relative errors up to 10%, rarely up to 20%, have been observed.

We have also studied the analysis of queueing networks with three-moment descriptors. The results show that the quality of the analysis strongly depend on the PH-fitting algorithm. The best results are obtained by the fitting algorithm using Minimal Acyclic PH distributions. However, in our experiments, the three-moment descriptors have not significantly improved the results for queueing networks with feedback in comparison to the two-moment descriptors used by FiFiQueues. Since three-moment descriptors considerably increase the runtime of the analysis, we currently refrain from using them in FiFiQueues.

Chapter 7

Closed Queueing Networks

Although most of this thesis concerns open queueing networks, many interesting systems can be elegantly modeled by closed networks. However, the progress in the analysis of closed networks has been much slower than for open networks. One reason for this is the fact that the bounded number of customers in a closed system prevents an intuitive decomposition.

Two popular examples illustrate the importance of closed network models. In the classical example shown in Figure 7.1 (adapted from [86]), a given number u of users share the CPU and some peripherals of a computer system. The users are modeled as jobs in the network and they continuously switch between "thinking" (at the u terminals) and "waiting" for the CPU and the peripherals. Closed network can also be used to model systems where the number of jobs is constant in the sense that a source creates a new job as soon as an old job has left the system. A popular example for such a scenario is the window-based flow control as implemented in TCP/IP. With this kind of flow control the number of jobs in the network with outstanding acknowledgment from the receiver is limited by the *window size* w. In the long run the average number of jobs in the system is equal to w. Such systems can be modeled by a closed network with a fixed number of jobs w.

In this chapter we present an approach that allows the analysis of closed queueing networks by decomposition. We achieve this by approximating such a closed queueing network by an open queueing network. The chapter is structured as follows. Section 7.1 gives a short overview of existing solution methods. We present our decomposition approach for closed networks in Section 7.2. In Sections 7.3 and 7.4 we describe an implementation of the approach and give some performance results. The chapter is summarized in Section 7.5.

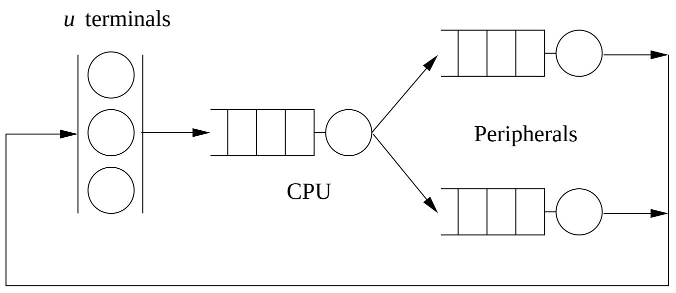


Figure 7.1: A time-sharing system as closed queueing network

7.1 Existing analysis methods

Gordon-Newell queueing networks

Gordon-Newell queueing networks (GNQN) [38] consist of n infinite M|M|1-FCFS queues with Markovian routing and no external arrivals nor departures. The constant number of jobs in the network is given by q. Since q is finite, it is possible to construct and to solve the underlying (finite) CTMC of the network. Such a direct solution is only feasible for small networks since the number of states in the CTMC suffers from the state space explosion phenomenon. In contrast, the *convolution algorithm* [19] significantly decreases the complexity of the CTMC solution to $O(n \cdot q)$ in time and to $O(q)$ in space by providing a recursive computation scheme.

The *Mean Value analysis (MVA)* [95] also avoids the direct solution of the CTMC. It is able to directly provide the measures of interest like mean queue length, etc., by using the so-called *arrival-theorem* [67, 107]. Again, the result is a recursive procedure with a complexity similar to the convolution algorithm: $O(n \cdot q)$ in time and $O(n)$ in space. Both methods provide exact results for GNQN. Various approximation methods have been developed for large values of q and n [81, 106, 117].

BCMP networks

The class of BCMP networks is the best-known class of mixed open and closed queueing networks with a product-form solution [8]. It allows routing based on job classes where each job class may either follow a closed or an open routing chain. All queueing stations can have load-dependent service rates and belong to one of the following types: M|M|m-FCFS, M|Cox|1 with processor sharing scheduling, M|Cox|∞, or M|Cox|1-LCFS with preemption.

BCMP networks can be solved by extended versions of the convolution algorithm and the MVA method [60]. However, they are much more complex than in the case of GNQNs. For example, the time complexity of the convolution method becomes $O(n \cdot r \cdot \prod_{i=1}^{r}(q_i + 1))$ where r is the number of classes with closed routes and q_i is the population size of class i.

The “functional approach”

This approximate approach [15] assumes that there exists a non-decreasing function f_i for each station i of a closed queueing network with

$$\mathrm{E}[N_i] = f_i(X_i), \tag{7.1}$$

where X_i is the throughput (or the utilization) of station i. Note that in a closed network with infinite queues, X_i can be expressed in terms of the throughput X of one reference station:

$$X_i = V_i X, \tag{7.2}$$

where V_i is the so-called visit-ratio of station i. Since the population size q is constant we have $\sum_{i=1}^{n} \mathrm{E}[N_i] = q$ which can be rewritten as

$$\sum_{i=1}^{n} f_i(X_i) = \sum_{i=1}^{n} f_i(V_i X) = q \tag{7.3}$$

using Equations (7.1) and (7.2). As can be seen, this equation is of the form $F(X) = q$ and can be solved by numerical methods (for example using a Newton-iteration). Once X is known, the other measures of interest can be computed.

Clearly, the key point is to know the expressions $f_i(X_i)$ for the queueing stations that occur in the network. Approximations of such functions can be found for the four types of queues of the BCMP approach and also for queues of type M|G|m-FCFS.

MEM for queueing networks

The authors of [62] have proposed a method for the analysis of arbitrary queueing networks with multiple servers and repetitive-service blocking (see Section 2.2.2) using the Maximum Entropy Method (MEM). The idea of MEM is to find the solution of the model that maximizes the entropy of the system under the condition that only the information given by the model specification is used.

The analyzed network may be open or closed and consists of n finite multiple-server queues of type GE|GE|m|K'; K where jobs can only leave the queue if the number of jobs in the queueing station is larger than K'. The complexity of the method is quite high. The solution algorithm consists of two stages that use iterative procedures. Stage 1 has a time order of about $O(c_1 n^6)$ in case all queues may block (i.e., their queueing capacity is smaller than the population size q). The complexity of stage 2 is $O(c_2 n^2 q^2)$, where c_1 and c_2 are the number of iterations in the successive stages.

Many other methods have been developed for the analysis of some special closed networks containing finite queues. They only support very restricted network topologies, like two-queue tandem networks, etc. We refer to [86] for an overview.

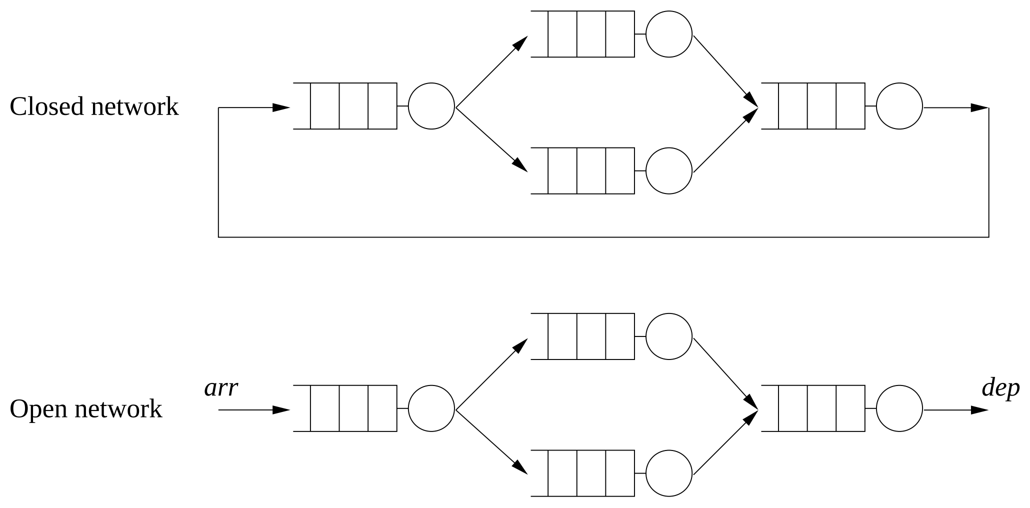


Figure 7.2: Closed network example and its corresponding open network obtained by cutting

7.2 Analysis of closed queueing networks by decomposition

We have presented in the previous chapters some very efficient methods for open queueing networks that follow the decomposition approach described in Chapter 2. Now, one may ask whether it is possible to also bring the efficiency and the flexibility of the decomposition approach to the class of closed networks. In this section we will describe a general procedure that is able to achieve this for certain network classes.

The idea is to transform a closed network into an open one by cutting one of its connections. This is shown for an example network in Figure 7.2. For the thus-obtained open network we have to find an external arrival descriptor *arr* in which

1. the external arrival descriptor *arr* is equal to the (resulting) descriptor *dep* of the traffic that leaves the network, and
2. the number of jobs in the network is equal to the specified population size q of the closed network.

We aim to find *arr* by applying the iteration procedure shown in Figure 7.3 to the closed network. The functions *err* and *err*$'$ are appropriate error functions and ε resp. δ the corresponding error bounds.

To implement this procedure we have to address two problems. First, at which connection should we place the cut in order to transform the closed network into an open one. Does its location influence the results? Secondly, how should we choose a new arrival descriptor *arr* inside the iteration? A possible solution of these problems is given in the next section where we describe an implementation of our approach

```
1  cut closed network and obtain open network
2  initialize $arr$
3  **loop**
4      analyze open QN and obtain external departure descriptor $dep$
5      **if** $err(arr, dep) > \varepsilon$ **or** $err'(\sum_{i=1}^{n} \mathrm{E}[N_i], q) > \delta$ **then**
6          choose new $arr$ based on the results of the network analysis
7      **else**
8          stop iteration
9      **end**
10 **end**
```

Figure 7.3: Analysis procedure for closed networks

that is based on the FiFiQueues method for open queueing networks. But we can already make the following observations:

- In Section 2.2.2 we have explained that the decomposition approach only supports communication blocking because no information about free queueing capacities can be exchanged between the queues. Since we are interested in closed queueing networks with fixed population size, we have to forbid job discarding. Hence, we will assume in the following that all queues have infinite capacity.

- Although the sketched procedure looks very simple its implementation is critical for complex network classes and traffic descriptors. It is unknown whether the iteration given above always terminates and whether more than one correct solution exist for a given closed network. Thus, the approach suffers from the same problem as other solution methods like MEM [62]. However, in our experiments (see below) it always terminated.

- The stopping condition $err'(\sum_{i=1}^{n} \mathrm{E}[N_i], q) > \delta$ is only an approximation to the original condition that the number of jobs in the closed network is equal to q. Except for the deterministic case, variations in the arrival and service processes may cause that the number of jobs in the open network vary around q, something that is not the case in a true closed queueing network.

7.3 Implementation with FiFiQueues

In this section we describe our implementation of the iteration scheme for closed queueing networks. We have chosen FiFiQueues (see Section 5.3) with its two-moment traffic descriptors as analysis method for the generated open networks. We have called the resulting analysis method *FiFiQueues Non-Blocking Closed*

(FiFiQueues-NBC). A first implementation of FiFiQueues-NBC has been presented in [94]. In the following sections we discuss an, especially for small q, improved implementation and give a deeper insight into its performance.

The model class of FiFiQueues-NBC is the model class of the original FiFiQueues adapted to closed networks without external arrivals and departures. Of course, one would wish to implement the method with more sophisticated descriptors but we have found that the two-moment traffic descriptors have special properties that allow a stable implementation of the iteration. We will describe them in the following. It is up to further research to investigate how more complex descriptors could be integrated.

First adaption

At the moment we ignore the question of the cut location and assume that the closed network is cut at an arbitrary connection. A first adaption of the iteration procedure to the two-moment descriptors looks as follows:

```
1  $\lambda_{arr,low} := 0$ ; $\lambda_{arr,high} := h$
2  $c^2_{dep} := 1$
3  do
4      $\lambda_{arr} := \frac{1}{2} \cdot (\lambda_{arr,high} + \lambda_{arr,low})$ ; $c^2_{arr} := c^2_{dep}$
5      call FiFiQueues and obtain external departure descriptor $(\lambda_{dep}, c^2_{dep})$
6      if $\sum_{i=1}^{n} \mathrm{E}[N_i] > q$ or network is unstable then
7          $\lambda_{arr,high} := \lambda_{arr}$
8      else
9          $\lambda_{arr,low} := \lambda_{arr}$
10     end
11 while $err(\lambda_{arr,low}, \lambda_{arr,high}) > \varepsilon$ or $err'(\sum_{i=1}^{n} \mathrm{E}[N_i], q) > \delta$
```

The algorithm is based on two assumptions. First, we assume that the number of jobs in the network q can be reached by an interval splitting iteration with the arrival rate λ_{arr}. The argument is similar to the one used in the functional approach (see Section 7.1). The initial value h in line 1 has to be set to an appropriate large value (a too large initial value only slows down the convergence — overloaded networks are avoided by the test in line 6). Note that we do not need to test λ_{arr} and λ_{dep} for equality since this is always fulfilled in networks without losses.

The second assumption concerns the squared coefficient of variation c^2. We have observed in the past that large queueing networks tend to "emboss" a network specific value for c^2 to the traffic stream. This means that the c^2 value of a traffic stream seems to depend only on the service processes and not on the c^2 value of the external arrival streams if the traffic stream has passed a sufficiently large number of queueing stations (provided that the utilization of the queueing stations is reasonably

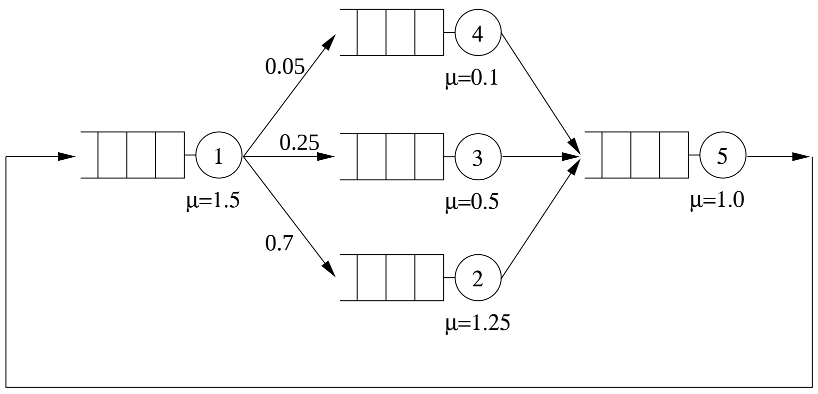


Figure 7.4: Example Gordon-Newell network

high). This is the reason why we have chosen an arbitrary initial value for c^2_{dep} in line 2 and simply assign c^2_{dep} to c^2_{arr} in line 4.

We test the algorithm at the example network shown in Figure 7.4. The figure shows the routing probabilities and the service rates of each node. All service processes are assumed to have $c^2_{service} = 1$, hence the network is a GNQN. The numerical results for $q = 50$ are shown in Table 7.1. The column labeled "decomposition" contains the results obtained by the decomposition approach by cutting the connection from node 5 to node 1. The column "MVA" shows the results of the MVA method (the c^2_N values have been calculated by simulation with relative 95%-confidence intervals smaller than 3%). Finally, the last column displays the error between the decomposition approach and the MVA/simulation relative to the latter.

The results are acceptable but they are a little bit disappointing. Remember that the corresponding open queueing network of the GNQN is a Jackson network which is exactly solvable by FiFiQueues. That means that all the relative errors are caused by our closed-network algorithm itself. The reason for the errors can be found when looking at the results of node 5 which is clearly the bottleneck of the network. Nearly all jobs in the network are waiting in the queue of this node while they quickly travel through the other queueing stations. As a consequence, the simulation result for c^2_N of node 5 (in the column titled "MVA") is near to 0 — we have a quite deterministic queue length distribution. In contrast, the decomposition approach provides a quite different value of 1.02 for c^2_N. This indicates that substantially more, or less than the required 50 jobs can be present in the corresponding open network.

Improved algorithm

We can better approach the behavior of the closed network at the bottleneck by the following modification of our algorithm [94]. We place the cut directly in front of the bottleneck, for example at the connection between node 2 and 5, and transform

node		decomposition	MVA	rel. error
	ρ	0.65	0.67	-3.0%
1	E[N]	1.88	2	-6.0%
	c_N^2	1.53	1.51	1.3%
	ρ	0.55	0.56	-1.8%
2	E[N]	1.21	1.27	-4.7%
	c_N^2	1.83	1.8	1.6%
	ρ	0.49	0.5	-2.0%
3	E[N]	0.96	1	-4.0%
	c_N^2	2.04	2.03	0.5%
	ρ	0.49	0.5	-2.0%
4	E[N]	0.96	1	-4.0%
	c_N^2	2.04	1.96	4.1%
	ρ	0.98	1.0	-2.0%
5	E[N]	45.1	44.7	0.9%
	c_N^2	1.02	0.01	10100%

Table 7.1: Numerical results for the example GNQN

node 5 into a queueing station with finite capacity q. When the bottleneck node experiences a high load and, hence, most of the jobs are waiting in the queue of the bottleneck node, this finite capacity limits the maximum number of jobs in the network and leads to a more deterministic queue length distribution at the bottleneck.

The complete modified algorithm for arbitrary network topologies is shown in Figure 7.5. Our experiments have shown that we can select an arbitrary connection to the bottleneck in line 2 if more than one connection exists. Similarly, if more than one bottleneck exists, an arbitrary one is chosen in line 1. The bottleneck of a closed queueing network with n queueing stations can be determined by solving the traffic equations [46]:

$$X_j = \sum_{i=1}^{n} X_i r_{ij},$$

where X_j denotes the throughput through node j and r_{ij} is the routing probability from node i to node j. The values of X_j can only be calculated relative to some other X_i $(i \neq j)$ since this system of equations is of rank $n-1$. If we select node 1 as reference node we can restate the traffic equations as follows:

$$V_j = \sum_{i=1}^{n} V_i r_{ij} = V_1 r_{1j} + \sum_{i=2}^{n} V_i r_{ij} = r_{1j} + \sum_{i=2}^{n} V_i r_{ij},$$

where the so-called visit ratio $V_j = X_j/X_1$ expresses the throughput of node j relative to node 1. This system of equations has a unique solution. In analogy

```
1  Determine bottleneck node $b$ of closed network
2  Cut connection to $b$ and obtain open network
3  Limit capacity of $b$ to $q$
4  $\lambda_{arr,low} := 0$ ; $\lambda_{arr,high} := h$
5  $c^2_{dep} := 1$
6  do
7      $\lambda_{arr} := \frac{1}{2} \cdot (\lambda_{arr,high} + \lambda_{arr,low})$ ; $c^2_{arr} := c^2_{dep}$
8      call FiFiQueues to obtain external departure descriptor $(\lambda_{dep}, c^2_{dep})$
9      if $\sum_{i=1}^{n} \mathrm{E}[N_i] > q$ or network is unstable then
10         $\lambda_{arr,high} := \lambda_{arr}$
11     else
12         $\lambda_{arr,low} := \lambda_{arr}$
13     end
14 while $err(\lambda_{arr,low}, \lambda_{arr,high}) > \varepsilon$ or $err'(\sum_{i=1}^{n} \mathrm{E}[N_i], q) > \delta$
```

Figure 7.5: Analysis procedure for closed networks based on FiFiQueues

to the utilization ρ_i in open queueing networks, we can now calculate the ratio $D_i = V_i/\mu_i$ for each node i. Then the bottleneck is the node i with the highest value of D_i.

Note that the initial value h of $\lambda_{arr,high}$ (line 4) must be sufficiently high in order to obtain a load of 1 at the bottleneck station. If the bottleneck has only one incoming edge, h must be at least twice the service rate of the bottleneck due to the factor of $\frac{1}{2}$ in line 7. Our experiments suggest to use a slightly larger value of 2.5 in order to compensate the losses at the station.

Table 7.2 states the results of the improved algorithm for the example GNQN. This time the relative errors are much smaller. Note that the large relative error of node 5's c^2_N is caused by the involved small absolute numbers. The other relative errors are within the 95%-confidence intervals of the simulation.

Complexity

The improved algorithm consists of two iterations of which the step count is usually not known in advance. The inner iteration is part of the FiFiQueues algorithm and some of its properties have been discussed in Chapter 6. In each inner iteration all queueing stations are analyzed with the complexities described in Section 5.3.8. Note that only the bottleneck station is modeled as a finite queueing station (of size q) and, hence, the time complexity of its analysis depends on the population q. Concerning the outer iteration, we have observed that there is no direct dependency on the population q (see Section 7.4.3 for a detailed example). Our experiments have shown that even for large networks and populations the required number of outer iterations usually stays below 30.

node		decomposition	MVA	rel. error
	ρ	0.67	0.67	0.0%
1	E[N]	2	2	0.0%
	c_N^2	1.5	1.51	-0.7%
	ρ	0.56	0.56	0.0%
2	E[N]	1.27	1.27	0.0%
	c_N^2	1.79	1.8	0.6%
	ρ	0.5	0.5	0.0%
3	E[N]	1	1	0.0%
	c_N^2	2	2.03	-1.5%
	ρ	0.5	0.5	0.0%
4	E[N]	1	1	0.0%
	c_N^2	2	1.96	2.0%
	ρ	1	1	0.0%
5	E[N]	44.7	44.7	0.0%
	c_N^2	0.02	0.01	100%

Table 7.2: Numerical results of the improved algorithm for the example GNQN

In addition to the iterations, the improved algorithm has to identify the bottleneck of the network. The solution of the system of traffic equations has a time complexity of $O(n^3)$ if a direct solution method like Gaussian elimination is employed, but reduces to $O(c \cdot n)$ in practice when a sparse storage and an iterative solver such as Gauss-Seidel are used (where c is the average number of outgoing connections per station).

7.4 Validation

In this section we examine the performance of our decomposition-based method for closed queueing networks, using four typical examples.

7.4.1 Queues in series

We start with a simple closed network that consists of three queues in series as shown in Figure 7.6 [94]. All service times are hyper-exponentially distributed with $c_{service}^2 = 2$. This network does not require any traffic merging or splitting, so that the corresponding open network can be analyzed by FiFiQueues without too much error.

Table 7.3 gives the results of the decomposition method in comparison to simulation for different service rates. The population size was set to 20. Again, the last column gives the relative errors. All relative 95%-confidence intervals of the

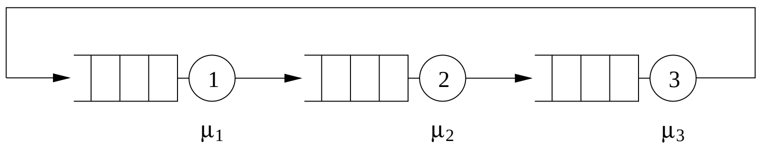


Figure 7.6: Three queues in series

simulation were below 1%.

The table shows that the algorithm does best when a distinct bottleneck is present in the network, i.e., in case $\mu_1 = \mu_3 = 1, \mu_2 = 0.5$. Then our "trick" with the finite queue provides very good results. Even when two stations have similar service rates ($\mu_1 = 1, \mu_2 = 2, \mu_3 = 1.1$), good results are obtained. The errors are, however, larger in cases where more than one bottleneck exist. Since the algorithm can select only one node as bottleneck it is not able to distribute the jobs evenly over all nodes in case all service rates are equal ($\mu_1 = \mu_2 = \mu_3 = 1$). The worst (but still okay) results are obtained when the network consists of two bottlenecks and one fast service station ($\mu_1 = \mu_3 = 1, \mu_2 = 2$); again, the algorithm can select only one node as bottleneck which results in different average queue lengths for node 1 and node 3 whereas the simulation indicates that both queue lengths should be equal.

The next experiment uses the same queueing network with $\mu_1 = \mu_3 = 1, \mu_2 = 0.5$, but this time the population is varied between 5 and 60. The results are shown in Table 7.4. As can be seen, the relative errors are larger for small population sizes. Similar results have been obtained for other networks. The explanation for this behavior is that the small number of jobs in the network do form traffic processes that are clearly not renewal processes. This fact contradicts with FiFiQueues' assumptions about the traffic processes and consequently, bad results are returned.

7.4.2 Queues with merging and splitting

With our next two networks we test how well the algorithm can handle more complex topologies with traffic merging and splitting. The networks are shown in Figure 7.7, respectively Figure 7.8. The obtained results for $q = 20$ can be seen in Table 7.5, respectively Table 7.6.

These examples illustrate that the algorithm for closed networks can only be as good as the underlying method for the open networks. Although q is not very small here, the errors are larger than in the case of three queues in series (see previous section) because FiFiQueues employs approximations to perform the traffic merging and splitting.

One distinct bottleneck: $\mu_1 = \mu_3 = 1, \mu_2 = 0.5$				
node		decomposition	simulation	rel. error
1	ρ	0.5	0.5	0.0%
	E[N]	1.5	1.55	-3.2%
2	ρ	1.0	1.0	0.0%
	E[N]	17.0	17.0	0.0%
3	ρ	0.5	0.5	0.0%
	E[N]	1.5	1.49	0.7%

One bottleneck: $\mu_1 = 1, \mu_2 = 2, \mu_3 = 1.1$				
node		decomposition	simulation	rel. error
1	ρ	0.95	0.95	0.0%
	E[N]	11.90	11.17	6.5%
2	ρ	0.48	0.47	2.1%
	E[N]	1.34	1.32	1.5%
3	ρ	0.84	0.86	-2.3%
	E[N]	7.76	7.51	3.3%

Three bottlenecks: $\mu_1 = \mu_2 = \mu_3 = 1$				
node		decomposition	simulation	rel. error
1	ρ	0.81	0.85	-4.7%
	E[N]	5.98	6.64	-9.9%
2	ρ	0.86	0.85	1.2%
	E[N]	7.45	6.66	11.9%
3	ρ	0.83	0.85	-2.4%
	E[N]	6.57	6.69	-1.8%

Two bottlenecks: $\mu_1 = \mu_3 = 1, \mu_2 = 2$				
node		decomposition	simulation	rel. error
1	ρ	0.88	0.91	-3.3%
	E[N]	8.25	9.36	-11.9%
2	ρ	0.44	0.45	-2.2%
	E[N]	1.12	1.22	-8.2%
3	ρ	0.93	0.91	-1.1%
	E[N]	10.63	9.42	12.8%

Table 7.3: Numerical results for three queues in series for different service rates

$q = 5$				
node		decomposition	simulation	rel. error
1	ρ	0.41	0.44	-6.8%
	E[N]	0.84	0.93	-9.7%
2	ρ	0.89	0.89	0.0%
	E[N]	3.27	3.14	4.1%
3	ρ	0.43	0.44	-2.3%
	E[N]	0.89	0.92	-3.3%

$q = 10$				
node		decomposition	simulation	rel. error
1	ρ	0.48	0.49	-2.0%
	E[N]	1.28	1.35	-5.2%
2	ρ	0.97	0.97	0.0%
	E[N]	7.42	7.34	1.1%
3	ρ	0.48	0.49	-2.0%
	E[N]	1.30	1.31	-0.8%

$q = 30$				
node		decomposition	simulation	rel. error
1	ρ	0.5	0.5	0.0%
	E[N]	1.50	1.57	-4.5%
2	ρ	1.0	1.0	0.0%
	E[N]	27.0	26.9	0.4%
3	ρ	0.5	0.5	0.0%
	E[N]	1.50	1.50	0.0%

$q = 60$				
node		decomposition	simulation	rel. error
1	ρ	0.5	0.5	0.0%
	E[N]	1.50	1.57	-4.5%
2	ρ	1.0	1.0	0.0%
	E[N]	57.0	56.9	0.2%
3	ρ	0.5	0.5	0.0%
	E[N]	1.51	1.50	0.7%

Table 7.4: Numerical results for three queues in series for various population sizes

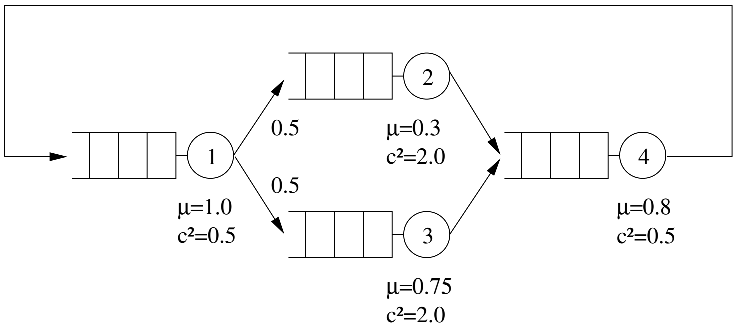


Figure 7.7: Network 1 with merging and splitting

node		decomposition	simulation	rel. error
1	E[N]	1.20	1.17	2.6%
2	E[N]	15.30	14.73	3.9%
3	E[N]	0.76	0.76	0.0%
4	E[N]	2.78	3.34	-16.8%

Table 7.5: Numerical results for network 1 with merging and splitting

7.4.3 Complex network example

Finally, we consider the more complex test network that is shown in Figure 7.9. The results for populations between 5 and 60 can be found in Table 7.7 and 7.8. As observed before in Section 7.4.1, the relative errors are largest for small population sizes.

It is worth to emphasize the fact the algorithm provides the best results for large populations, i.e., when the direct solution of the CTMC underlying the network is not easy due to the size of the CTMC. The number of states s of the CTMC

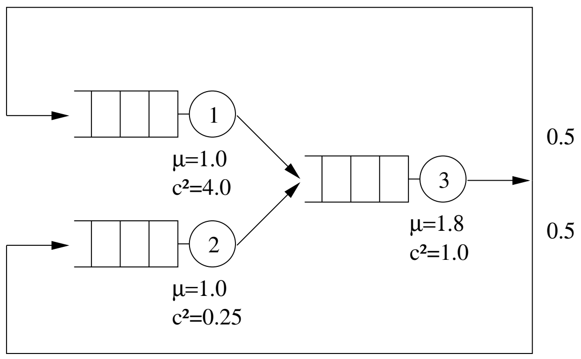


Figure 7.8: Network 2 with merging and splitting

node		decomposition	simulation	rel. error
1	E[N]	6.40	6.76	-5.3%
2	E[N]	3.64	3.66	-0.6%
3	E[N]	9.96	9.58	4.0%

Table 7.6: Numerical results for network 2 with merging and splitting

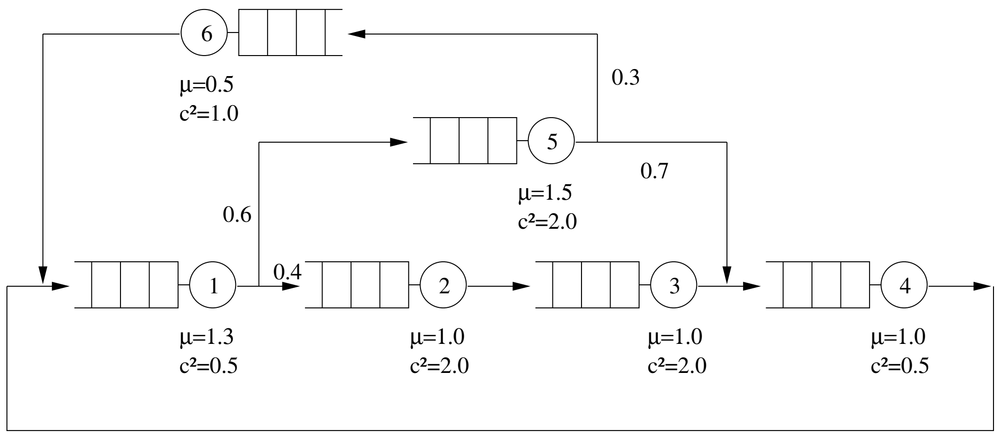


Figure 7.9: Complex closed network

underlying a Gordon-Newell network is given by

$$s = \begin{pmatrix} n+q-1 \\ n-1 \end{pmatrix},$$

where n is the number of queueing stations and q is the population size [46]. For networks with phase-type service time distributions, the number of states for large q is approximately given by

$$s \approx \begin{pmatrix} n+q-1 \\ n-1 \end{pmatrix} \cdot \prod_{i=1}^{n} p_i,$$

where p_i is the number of phases of the service time distribution of station i. Hence, the underlying CTMC of the complex network example with $n = 6$ and $q = 30$ would comprise approximately $2 \cdot 10^8$ states, whereas the largest CTMC constructed by FiFiQueues during the analysis of the same network has around 240 states.

Table 7.9 shows the runtimes (in seconds) of the FiFiQueues algorithm and of the discrete-event simulation for the evaluation of the network with different populations. For FiFiQueues, we have recorded the runtimes for two different implementations of the finite queue analysis. As expected, the runtime of the implementation based on the Gauss-Seidel iteration quickly increases with the population because the bottleneck station is modeled as a finite capacity station. The runtime of the implementation based on the Cyclic Reduction method (see Section 3.4.3) is mostly

$q = 5$				
node		decomposition	simulation	rel. error
1	ρ	0.65	0.69	-5.8%
	E[N]	1.28	1.27	0.8%
2	ρ	0.34	0.36	-5.6%
	E[N]	0.54	0.57	-5.3%
3	ρ	0.33	0.36	-8.3%
	E[N]	0.56	0.59	-5.1%
4	ρ	0.71	0.74	-4.1%
	E[N]	1.65	1.57	5.1%
5	ρ	0.34	0.36	-5.6%
	E[N]	0.54	0.56	-3.6%
6	ρ	0.30	0.33	-9.1%
	E[N]	0.43	0.45	-4.4%

$q = 10$				
node		decomposition	simulation	rel. error
1	ρ	0.82	0.85	-3.5%
	E[N]	2.69	2.71	-0.7%
2	ρ	0.42	0.44	-4.5%
	E[N]	0.84	0.87	-3.4%
3	ρ	0.42	0.44	-4.5%
	E[N]	0.90	0.94	-4.2%
4	ρ	0.88	0.91	-3.2%
	E[N]	4.14	3.98	4.0%
5	ρ	0.43	0.44	-2.3%
	E[N]	0.82	0.85	-3.5%
6	ρ	0.38	0.40	-5.0%
	E[N]	0.62	0.65	-4.6%

Table 7.7: Numerical results for the complex network ($q = 5$ and $q = 10$)

$q = 30$				
node		decomposition	simulation	rel. error
1	ρ	0.93	0.93	0.0%
	E[N]	6.87	7.11	-3.4%
2	ρ	0.48	0.49	-2.0%
	E[N]	1.07	1.10	-2.7%
3	ρ	0.48	0.49	-2.0%
	E[N]	1.18	1.21	-2.5%
4	ρ	0.99	1.00	-1.0%
	E[N]	19.10	18.76	-1.8%
5	ρ	0.48	0.47	-2.1%
	E[N]	1.04	1.05	-1.0%
6	ρ	0.43	0.44	-2.3%
	E[N]	0.77	0.77	0.0%

$q = 60$				
node		decomposition	simulation	rel. error
1	ρ	0.94	0.94	0.0%
	E[N]	8.47	8.32	1.8%
2	ρ	0.49	0.49	0.0%
	E[N]	1.10	1.10	0.0%
3	ρ	0.49	0.49	0.0%
	E[N]	1.22	1.23	-0.8%
4	ρ	1.00	1.00	0.0%
	E[N]	47.36	47.48	-0.3%
5	ρ	0.49	0.49	0.0%
	E[N]	1.07	1.07	0.0%
6	ρ	0.44	0.44	0.0%
	E[N]	0.79	0.79	0.0%

Table 7.8: Numerical results for the complex network ($q = 30$ and $q = 60$)

	FiFiQueues		
q	Gauss-Seidel	Cyclic Red.	simulation
10	1	1	16
30	3	3	16
60	10	7	16
90	20	8	16
120	33	8	16
150	50	8	16

Table 7.9: Runtimes (in seconds) for the evaluation of the complex network

dominated by the administration overhead of the outer and inner iteration and hence only shows a weak dependency on the population. This method is much faster than the Gauss-Seidel iteration and the simulation. The latter exhibits nearly constant runtimes.

We finally comment on the convergence behavior of our new algorithm. For that purpose, Figure 7.10 shows for $q = 30$ how the algorithm modifies the arrival rate for the open network in each (outer) iteration in order to reach the desired number of jobs. The interval splitting algorithm first lowers the arrival rate to a fourth of the initial value, then the arrival rate is slowly increased (until iteration 6). The clear "dip" in the curves, hence, is an artifact of the interval splitting method; a more advanced method could probably avoid it. In this example the stop condition has been fulfilled after 14 iterations but we can see that a good approximation is already reached after about 10. Figure 7.11 again shows the number of jobs as function of the iteration number, this time for different population sizes. No direct dependency between the population and the number of required iterations can be observed.

7.5 Summary and conclusions

In this chapter we have proposed a simple decomposition-based method for the analysis of closed queueing networks. It is especially attractive because it is founded on existing analysis methods for open networks. Our implementation with FiFiQueues suggests that the method is able to provide useful results for a broad class of closed networks. Additionally, the method is very fast even for large networks and populations. But the experiments have also shown that

(a) it works best when the network contains exactly one bottleneck, and that

(b) it can only be as good as the method employed for the analysis of the generated open networks.

Concerning (b), it would be desirable to use more sophisticated traffic descriptors than the two-moment descriptors of FiFiQueues. More research has to be done in

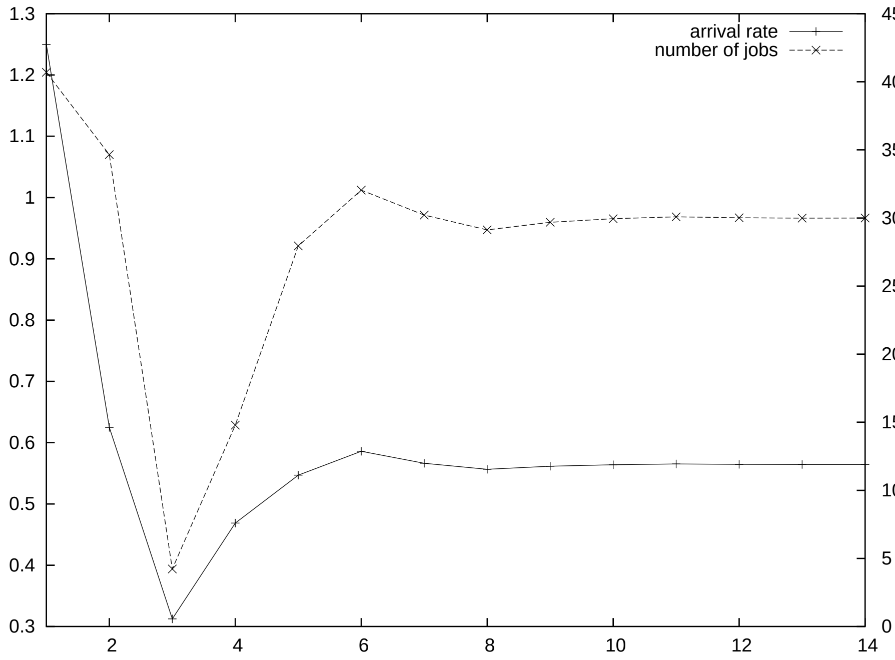


Figure 7.10: Arrival rate and number of jobs as function of the iteration number ($q = 30$)

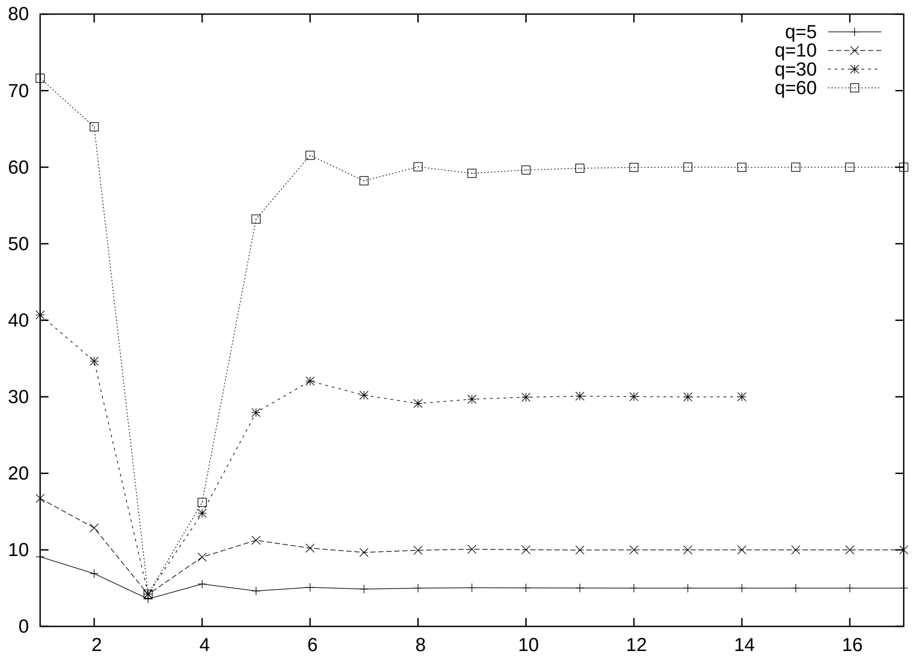


Figure 7.11: Number of jobs as function of the iteration number and the population

this area but it is to be expected that this requires a procedure for the estimation of the correct arrival descriptor that is much more complex than the one presented in Figure 7.5.

Chapter 8

Tool Support

The FiFiQueues approach and its extension to closed networks have proven to be stable and reliable enough for end users. In this chapter we present an integrated tool environment that allows an easy access to the underlying algorithms. The tool also contains a simulator for the steady-state simulation of queueing networks.

We give a short description of this environment, which we regard as the reference implementation of the FiFiQueues approach, in Section 8.1. Additionally, we briefly discuss a port of FiFiQueues into the Möbius Framework in Section 8.2. Two optimizations for the simulation of queueing networks are discussed in Section 8.3. Finally, we give a summary of the chapter in Section 8.4.

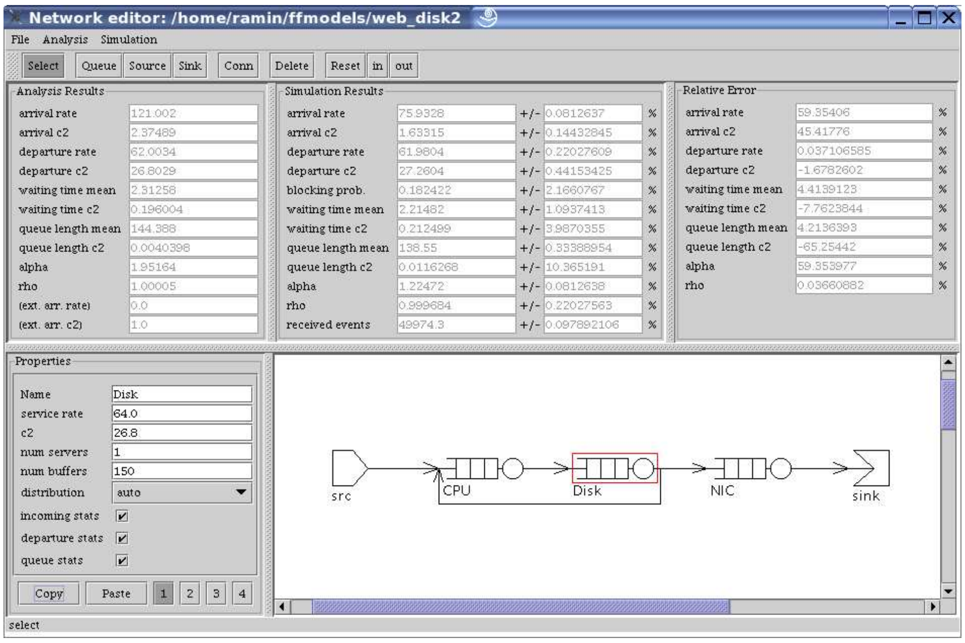
8.1 The FiFiQueues network designer

The FiFiQueues network designer consists of a graphical user interface written in Java, the numerical analysis module, and the simulation module. The latter two have been written in C++.

The graphical user interface

The graphical user interface allows to construct, edit and study open and closed queueing networks of arbitrary topology. The networks can be evaluated by numerical analysis or by simulation. Figure 8.1 shows a screenshot of the main window. The lower part of the window shows the edited network and the properties of the currently selected node. The upper part displays the results of the numerical analysis (left section) and the results of the simulation (middle section, including the 95% confidence intervals) as well as a comparison of both methods (right section).

Every object in the network has properties that can be edited via the user interface. Figure 8.2(a) shows the properties of a finite queueing station while the user is selecting a service distribution. The global-properties panel (see Figure 8.2(b))



Analysis Results

arrival rate	121.002
arrival c2	2.37489
departure rate	62.0034
departure c2	26.8029
waiting time mean	2.31258
waiting time c2	0.196004
queue length mean	144.388
queue length c2	0.0040398
alpha	1.95164
rho	1.00005
(ext. arr. rate)	0.0
(ext. arr. c2)	1.0

Simulation Results

arrival rate	75.9328	+/-	0.0812637	%
arrival c2	1.63315	+/-	0.14432845	%
departure rate	61.9804	+/-	0.22027609	%
departure c2	27.2604	+/-	0.44153425	%
blocking prob.	0.182422	+/-	2.1660767	%
waiting time mean	2.21482	+/-	1.0937413	%
waiting time c2	0.212499	+/-	3.9870355	%
queue length mean	138.55	+/-	0.33388954	%
queue length c2	0.0116268	+/-	10.365191	%
alpha	1.22472	+/-	0.0812638	%
rho	0.999684	+/-	0.22027563	%
received events	49974.3	+/-	0.097892106	%

Relative Error

arrival rate	59.35406	%
arrival c2	45.41776	%
departure rate	0.037106585	%
departure c2	-1.6782602	%
waiting time mean	4.4139123	%
waiting time c2	-7.7623844	%
queue length mean	4.2136393	%
queue length c2	-65.25442	%
alpha	59.353977	%
rho	0.03660882	%

Properties

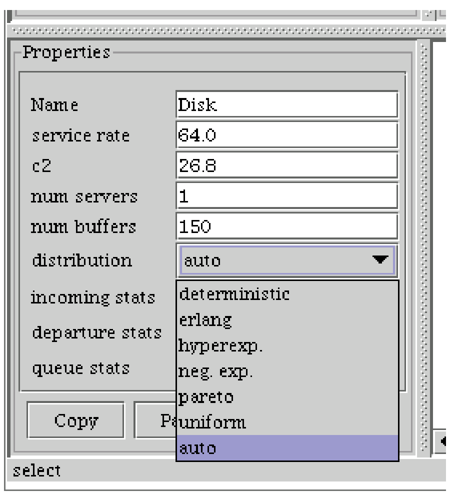
Name	Disk
service rate	64.0
c2	26.8
num servers	1
num buffers	150
distribution	auto
incoming stats	✓
departure stats	✓
queue stats	✓

Figure 8.1: Main window of the graphical user interface

allows to control the length of the simulation and the parameters specific to closed networks.

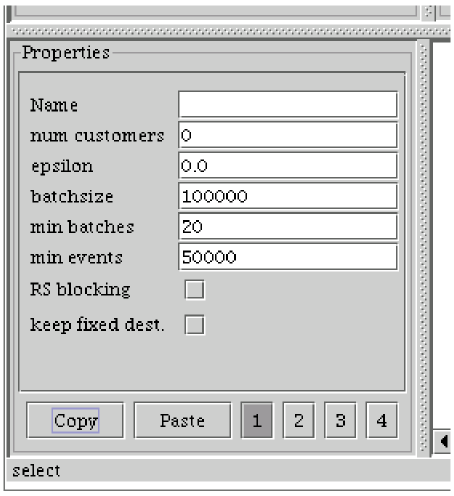
The user interface communicates with the numerical analyzer and the simulator via text files. As an example, the network shown in the screenshot is translated into the following textual description in order to evaluate it.

```
# Queue mapping
# 0 CPU
# 1 NIC
# 2 Disk
network_props
1 3 1 0 100000 20 50000 0  0   0.0
source_props
90.0 1.2 0 6
queue_props
1200.0 26.8 1 150 6 1  1  1
1401.0 26.8 1 150 6 1  1  1
64.0 26.8 1 150 6 1  1  1
counter_dest
-1
r
0.0 0.0 1.0 0.0
0.0 0.0 0.0 1.0
```


Properties

Name: Disk
service rate: 64.0
c2: 26.8
num servers: 1
num buffers: 150
distribution: auto
incoming stats
departure stats
queue stats

(a) Properties of a network node


Properties

Name:
num customers: 0
epsilon: 0.0
batchsize: 100000
min batches: 20
min events: 50000
RS blocking: ☐
keep fixed dest.: ☐

(b) Global properties of the network

Figure 8.2: Property editor

```
0.5 0.5 0.0 0.0
b
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
```

The numerical analysis module

The numerical analysis module is the core of the implementation. It incorporates the FiFiQueues algorithm (see Chapter 5) and the extension for closed queueing networks as described in Chapter 7.

The simulation module

The simulation module offers the discrete-event simulation of open and closed queueing networks. It is described in Section 8.3 and in Appendix B.

8.2 Integrating FiFiQueues in the Möbius Framework

In [56] it is examined how the FiFiQueues algorithm can be integrated into the Möbius framework. Möbius [23, 26] is a modeling software developed by the PERFORM research group which supports different modeling formalisms and solvers

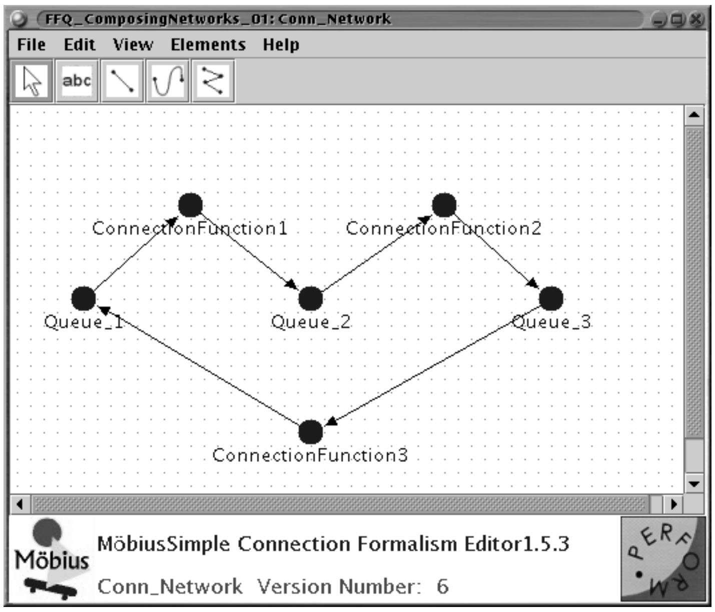

FFQ_ComposingNetworks_01: Conn_Network
File Edit View Elements Help
abc
ConnectionFunction1
ConnectionFunction2
Queue_1
Queue_2
Queue_3
ConnectionFunction3
Möbius
MöbiusSimple Connection Formalism Editor1.5.3
Conn_Network Version Number: 6
PERFORM

Figure 8.3: Möbius' connection editor (from [56])

while offering a common interface allowing tool modules to exchange information. Using this interface it is possible to build and solve models out of components that are based on different formalisms. Möbius allows to analyze composed models in two different ways: (i) by generating and analyzing the complete flat state space of the whole model, or (ii) by a fixed-point iteration, similarly to the iteration scheme of FiFiQueues. Models that are intended for the latter analysis approach are constructed with the connection editor (see Figure 8.3).

The integration of FiFiQueues required to extend the Möbius tool since it originally supports only finite state spaces. Möbius performs the network analysis using variables and actions that are associated to the states of the (finite) state spaces of the model components (the so-called atomic models). With infinite state spaces, a much tighter connection between the variables and the atomic models is required since the single states of the infinite state spaces cannot be explicitly constructed in finite time. This is achieved by special reward functions that "tunnel" information directly from the atomic model to the higher levels. Figure 8.4 sketches the resulting communication between the tool levels.

8.3 Steady-state simulation of queueing networks

The simulator of the FiFiQueues network designer offers the discrete-event, steady-state simulation of queueing networks. It is based on the method of independent replications [92] since our experiments have shown that the most efficient method to speed up simulation, without changing the simulator's core as required in more

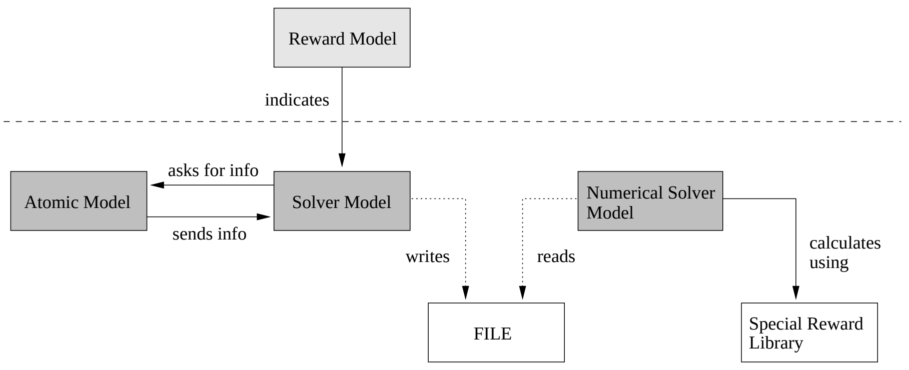


Figure 8.4: Special rewards in Möbius (from [56])

complex methods like rare event simulation [39, 112], is to simulate the replications in parallel.

In this section, we will discuss two optimizations that are specific to the simulation of queueing networks. First, we discuss the management of the future-event set in Section 8.3.1. An approach to shorten the initial transient phase of the simulation is presented in Section 8.3.2.

8.3.1 Managing the future-event set

The future-event set (FES) is one of the central components of discrete-event simulations. When a component of the simulated model creates an event with a scheduled time located in the future, this event has to be inserted into the FES. As soon as the virtual clock of the simulation reaches the activation time associated with the event, it is removed from the FES and scheduled for execution. Since the events stored in the FES have to be sorted by activation time, every access to the FES may lead to considerable CPU overhead.

Thus, it is not surprising that the implementation of the FES was an important research topic in the area of discrete-event simulations. The simplest implementations are based on single and multiple doubly linked linear lists or on doubly linked binary trees. Alternatives, like Henriksen's approach, combine linear lists and binary trees to obtain better performance [36]. Today, sophisticated data structures like Calendar Queues [17] provide nearly optimal results, i.e., $O(1)$, at least for non-distributed simulations.

In this thesis, we are only interested in simulation of queueing networks. Naturally, this choice affects the characteristics of the FES. We discuss these characteristics as well as a simple optimization to the FES management in the following. We assume that our model to evaluate is an open queueing network with n queueing stations. The usual notation is employed, e.g., $\mathrm{E}[W_i]$ stands for the mean waiting time at queueing station i.

Characteristics of the FES for queueing networks

An open queuing network model contains only two sources of events for the FES:

1. Every external arrival process continuously holds an event in the FES. Usually, this event represents the arrival of a job into the network. As soon as the event has been processed (i.e., its activation time has been reached), a new event is put into the FES for the next arrival.

2. Similarly, every *busy* service station continuously generates one event which represents the end of service time.

Hence, the minimum FES size $\min(f)$ and the maximum FES size $\max(f)$ simply are

$$\min(f) = n_{ext}, \tag{8.1}$$
$$\max(f) = n_{ext} + n, \tag{8.2}$$

where n_{ext} is the number of external arrival processes. The mean number $\bar{f}_t$ of events in the FES *over virtual simulation time* is given by

$$\bar{f}_t = n_{ext} + \sum_{i=1}^{n} \rho_i,$$

where ρ_i is the utilization of node i, i.e., the probability that node i is busy. For FES implementation purposes we are interested in the FES size $\bar{f}$ *at insertion and deletion times.* It cannot be computed exactly but Equation (8.2) implies that even queueing network models that are regarded as large today, i.e., $n = 50$, result in very small $\max(f)$ compared to other simulation scenarios. From this fact we can deduce that a FES implementation suitable for queueing networks should rely on data structures with very small overhead if the goal is to outperform a simple linear list implementation.

In an open queueing network, a job, after being generated by an external arrival process, passes once or more through some of the network nodes and finally disappears when it leaves the network. The birth of the job, as well as each time the job is served by a node, will cause the simulation system to insert an event into the FES. When the simulation stops the FES still contains some unprocessed events. In the following, we will ignore those remaining events since their number can usually be neglected when compared to the huge number of processed events. So, we assume that the number of events inserted into the FES is (nearly) equal to the number of processed events.

The mean number of events generated in a given time interval Δt can be computed as follows:

1. Each external arrival process with arrival rate λ_{ext} generates $\Delta t \cdot \lambda_{ext}$ jobs in time interval Δt, where each job leads to an event inserted into the FES.
2. Each node i in the network serves $\Delta t \cdot \mu_i \rho_i$ jobs in time interval Δt. If no losses occur, this expression reduces to $\Delta t \cdot \lambda_{A,i}$. Again, each service causes a new event in the FES.

Then the total number of FES insertion operations in Δt is

$$\Delta t \cdot \left(\sum_{i=1}^{n_{ext}} \lambda_{ext,i} + \sum_{i=1}^{n} \mu_i \rho_i \right) .$$

When losses are possible, the ρ_i are rarely known in advance, so this equation will only give an approximation in this situation.

FES management

We stated that, due to the small size, any FES implementation that relies on too complex data structures would be outperformed by a simple linear list implementation with linear search. A simple way to improve a straight linear search through the list would be available if we would knew at least the average insertion position in advance: we could decide at which end of the list the search should be started. This would improve the performance even for small $\bar{f}$. Note that for this purpose we are interested in the average *normalized* insertion position, i.e., the average of the insertion position divided by the size of the FES at time of insertion. If it is smaller than 0.5 we start at the head, otherwise the search is started at the tail of the list.

If we assume that the nodes of the queueing network are independent, an estimation of the insertion positions can be obtained by comparing the service rates of the queueing stations. A station with large service rate generally has a higher probability to place new events more toward the tail (which represents the “present”) of the FES. However, the correlations between the nodes are generally too strong to provide useful results in practice. Hence, our implementation simply measures the average insertion position for each node once after the warm-up phase and keeps it till the end of the simulation. This assumes that there are no long-term changes in the behavior of the model after the warm-up phase that influence the insertion position.

Unfortunately, our experiments have shown that this approach does not provide a noticeable speed-up when applied to general queueing networks. For example, all the stations of the 9-nodes network introduced in Section 6.1.2 have very similar normalized insertion positions near to 0.5. Interestingly, the optimization yields the maximum possible speed-up for a model that is quite common: the multiplexer (Figure 8.5). In this case the model consists of an external arrival process with high

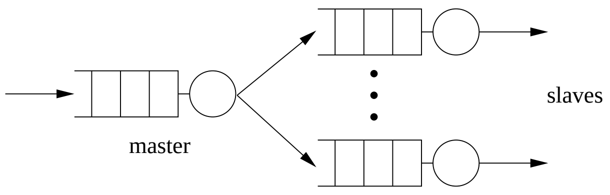


Figure 8.5: Best case scenario for the FES optimization

component	normalized insertion position
external source	0.11
master station	0.18
slave station 1	0.61
...	
slave station 7	0.61

Table 8.1: Average normalized insertion positions for the best case scenario

arrival rate (and, hence, low event generation interval) and several queueing stations with service rates that are small compared to the external arrival rate.

We test this scenario with an external source with rate 6.3 and SCV 2.0. The multiplexing master queueing station has a service rate of 9.0 and a service time SCV of 0.5. This station evenly distributes the served jobs to seven slave queueing stations with service rate 1 and SCV 0.5. Hence, the slave stations experience a load of 90%. We can observe that one third of the events in the FES is generated by the external source, one third is generated by the service process of the master station, and the last third is caused by the slave queues. Because of the difference between the service rates of the master station and the slave stations, quite distinct average normalized insertion positions arise (see Table 8.1). Nevertheless, the measured speed-up of the new FES management algorithm is low; the number of processed events per second only increases by 2%-3% compared to the simple linear search, depending on the amount of non-FES-related tasks during the simulation, e.g., statistical data collection. The speed-up slightly increases with the number of slave stations.

8.3.2 The initial transient phase

As any dynamic system, a queueing process with random arrival and service processes is in a transient phase after initialization. During this initial transient phase its characteristics vary with time. After a period of time the system approaches its steady-state, or statistical equilibrium, and measures like mean queue length take on their steady-state values. For queueing stations with infinite buffer this steady-state only exists if the arrival rate is smaller that the service rate. For queueing networks this condition has to be fulfilled for every station with infinite buffer.

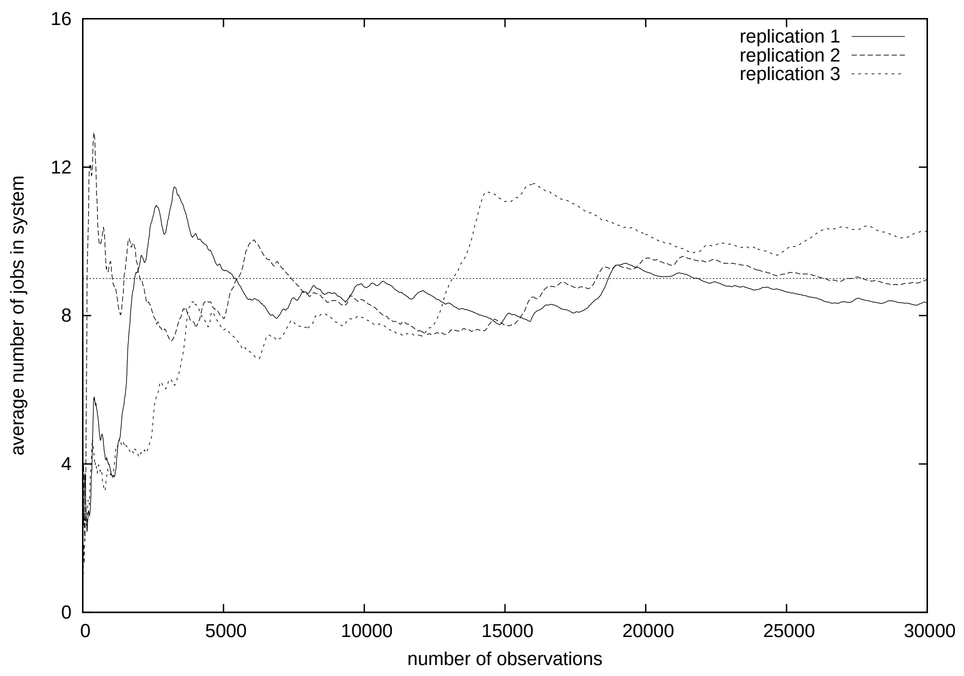


Figure 8.6: Progress of the running average number of jobs in system for three different replications

In general, the system asymptotically approaches its steady-state. An important question in this context is how to treat the initial transient phase with regard to the collection of statistical data since all observations made during this phase provide data that is not characteristic for the steady-state behavior of the system. These observations may slow down the convergence of the estimators toward the steady-state value. Hence, it seems to be natural to delete all observations made during the initial transient phase. The efficiency of the deletion has been intensively discussed in the literature because, in general, it increases the variance of the estimator. However, in case of the method of independent replications the deletion is useful and clearly improves the results because for this method the initial transient phase has to be traversed in each replication.

Figure 8.6 visualizes how the average number of jobs in a M|M|1 queue with $\rho = 0.9$ is affected by the initial transient phase. The figure shows the progress of the average number of jobs with increasing number of observations as obtained by three replications. All replications start with an empty queue. The steady-state number of jobs is 9. As can be observed the transient phase seems to end at different times for each replication. For example, in replication 2 the average number begins to approach the steady-state value nearly directly after the begin of the simulation whereas the other two replications, especially replication 3, are more affected by the initial transient period.

Many methods have been developed to recognize or to predict the end of the transient phase (see [92] for an overview). The popular approach based on the observation of the collected data does not perform well with long-range dependent processes. In the last fifteen years the interest in long-range dependent processes has considerably increased [42]. Currently, no general method is known that is able to distinguish between fluctuations in the estimator caused by the initial transient period and fluctuations stemming from long-range changes in the arrival or service process as observed in systems with self-similar characteristics [93].

Hence, we use a fixed number of deletions in our simulator (usually, $n/10$, where n is the number of observations in one replication). Alternatively, we do not perform any deletions but try to shorten the length of the initial transient phase as described in the following section.

Shortening the initial transient phase

Much research has been undertaken to determine the optimal initial conditions for the simulation of queueing stations, in the sense that they would minimize the influence of the initial transient phase on the steady-state results. In the literature ambiguous conclusions have been drawn from experiments with different initial conditions. [77] shows that the mean square error of the mean queue length can reach its minimum value if the queue is initialized empty and idle. A different conclusion has been reached later by a deeper analysis of the transient behavior of queueing processes. In the case of the estimation of the mean queue length in M|M|1 queues, the authors of [1] indicated that the optimal initial state is when the queue length is initialized with about 1.5 times the steady-state mean. For example, in a M|M|1 queue with $\rho = 0.9$ and mean number of jobs in system of 9 the optimal initial value is 15.

No theoretical results for the optimal initial queue length are known for more complex queueing processes or entire queueing networks. Hence, it has to be estimated. Note that it is ill-advised to choose an arbitrary value that looks “large enough” as initial queue length since it was shown that starting from a state much larger than the steady-state mean can result in a very long transient period [92]. Hence, [92] summarized the research in this area:

> *“Thus, because in real situations the steady-state mean is unknown, it is much safer to initialise systems as empty and idle, [...]”.*

In the following we illustrate how numerical analysis can be used to shorten the initial transient phase. We will use the results provided by FiFiQueues to choose an initial queue length for the simulation of the queueing network shown in Figure 8.7. That example models a component of a communication network. A traffic stream arrives to a queueing station with finite buffer (node 1). Both the inter-arrival times and the service times at the queue have a low variance. However, losses are possible with a small probability due to the finite buffer. Each blocked job at queue 1 will

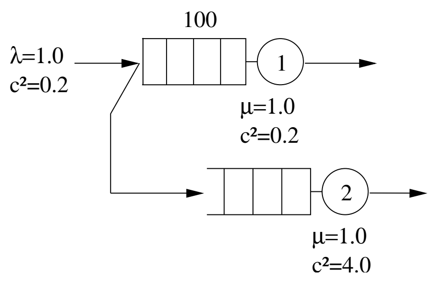


Figure 8.7: Queueing network with loss

generate a job for queue 2 (which models some recovery procedure initiated in case of a loss).

Although FiFiQueues is able to exactly analyze queue 1 and the inter-arrival time distribution for queue 2, important performance measures of queue 2 like its mean queue length are not well predicted. This is because the inter-arrival times of queue 2 are strongly correlated and do not form a renewal process. On the other hand, the simulation of this queueing network (with same arrival and services processes as for FiFiQueues) suffers from a long initial transient phase. Figure 8.8 shows a typical run of the simulator. The average number of jobs in queueing station 1 and the number of samples collected at queue 2 are plotted against the progress of the simulator (expressed in number of samples collected at queue 1). We can clearly recognize that the initial transient phase covers at least the first 30000 samples at queue 1. During this phase the second queue receives nearly no jobs. After that, the estimator of the average number of jobs starts to approach its mean value and the number of samples for queue 2 quickly raises.

What happens if we use the exact results of FiFiQueues for queue 1 to specify an initial queue length? FiFiQueues computes a mean queue length of approximately 50. Following [1], we choose $50 \times 1.5 = 75$ as initial queue length. We have first started 100 replications of the simulation without any deletion and with an initial queue length of 0. The stopping criterion was to collect 1000 samples for queue 2. Then we have started again 100 replications without deletion and with the same stopping criterion, but this time we have initialized queue 1 with 75 jobs.

As expected, the replications of the first run needed longer to collect the required 1000 samples for queue 2. On average, they simulated approximately $5.4 \cdot 10^5$ jobs before stopping whereas the replications of the second run stopped after $5 \cdot 10^5$ jobs. It is noteworthy that the difference of 40000 jobs covers a large part of the initial transient phase as shown in Figure 8.8. Finally, we want to emphasize that both runs have computed the correct mean queue length for queue 2 (with respect to the required confidence).

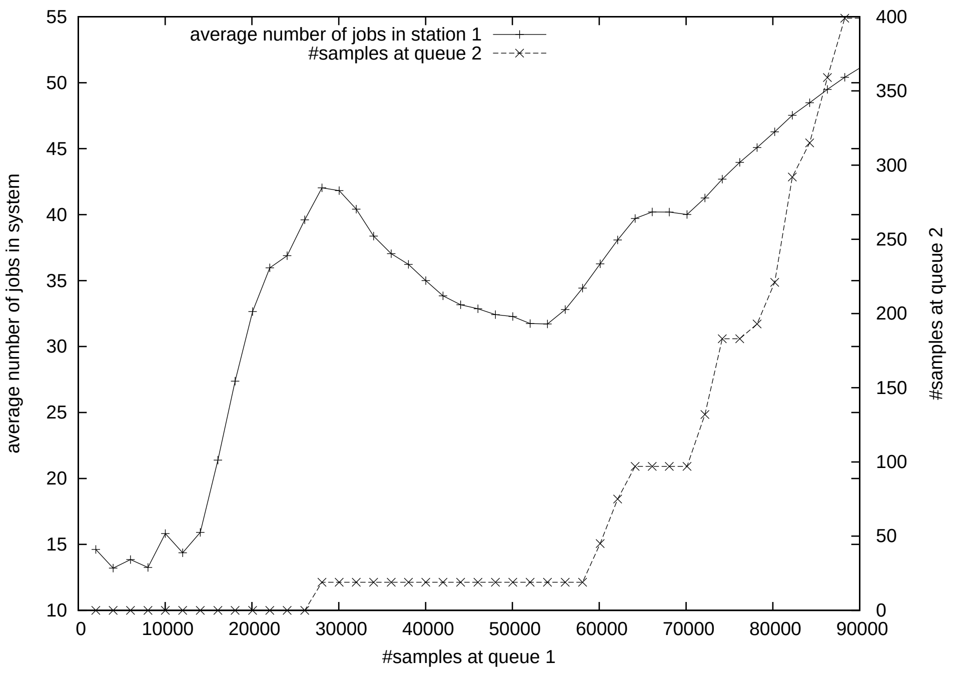


Figure 8.8: Transient behavior for sample network with loss

8.4 Summary and conclusions

In this chapter, we have presented the FiFiQueues network designer. It is an integrated tool environment and comprises a GUI written in Java, the implementation of the FiFiQueues algorithm for open queueing networks, its extension to closed networks, and a discrete-event simulator for the steady-simulation of queueing networks. Additionally, we have briefly discussed a port of FiFiQueues into the Möbius Framework.

Concerning the simulation, we have presented two optimization techniques for the simulation of queueing networks. The first technique aims to reduce the overhead of the FES management. It is easy to implement, but only provides a minor speedup. The other technique substantially shortens the length of the initial transient phase in a rare-event scenario. Currently, it is not used in the tool due to its experimental status, but first experiments have shown that it can reduce the effects of the transient phase by a reasonable initialization of the simulation.

Chapter 9

MAP-Based Traffic Descriptors

In the previous chapters we have examined the analysis of queueing networks using first-order traffic descriptors. Those traffic descriptors can be constructed in such a way that they approximate the inter-arrival distribution of the underlying traffic stream to any required precision (see Section 5.5). However, they fail whenever higher-order characteristics, e.g., autocorrelation, of the arrival process dominate or influence the performance of the queueing stations. To respect such scenarios more complex traffic descriptors are required. In this chapter we will examine the use of Markovian Arrival Processes (MAPs, see Section 3.1) as traffic descriptors in queueing network. The motivation for this choice is the fact that complex arrival patterns, for instance those appearing in communication networks, can be described well using MAPs.

The class of MAPs has the interesting property that it is closed under the three network operations *superposition*, *splitting*, and *service*. Thus, in theory there is no need of auxiliary steps that convert traffic descriptors into internal representations for the analysis operations and vice versa. Unfortunately, an analysis solely based on MAPs suffers under the well-known state space explosion phenomenon: the superposition step in queueing networks as well as the service operation at MAP|MAP|1|K queues produce MAPs with very large numbers of states. Additionally, the departure process of a MAP|MAP|1 queue is not a finite MAP. This problem is not new and has been recognized by other authors as well. In the past years various methods have been proposed that aim to reduce large MAPs to smaller ones. Most of them are approximations that try to preserve the most important aspects of the original traffic process at the cost of aspects that are considered to be less important for the performance evaluation of the queueing network. Generally, two kinds of techniques can be identified: first, methods that can be applied to any MAP and secondly, methods that have been especially designed for the departure process of the MAP|MAP|1(|K) queue. Although all proposed techniques work very well for a restricted set of queueing networks, no approach is known that is able to provide good results for arbitrary networks. We believe that currently the best way is to rely on hybrid analysis techniques: whenever a queueing station has to be analyzed, we first examine the involved processes and then select the technique that is known to provide the best results for the particular case. In this spirit, we contribute with new approximation and reduction methods in this chapter.

This chapter is organized as follows. In Section 9.1, we introduce MAPs as traffic descriptors in queueing networks and briefly discuss the state space explosion problem. In Section 9.2, we discuss how the size of a MAP can be reduced by removing equivalent states. In Section 9.3, we present a method that approximates the departure process of a MAP|MAP|1 queue by a finite MAP. In Section 9.4, we show how departure processes of MAP|MAP|1(|K) can be approximated by condensing levels of the underlying QBD. The method developed in Section 9.5 fits a MAP to given moments of the inter-arrival time and to a positive, exponentially decreasing autocovariance function. Section 9.6 describes how a MAP can be approximated by a renewal process. Finally, Section 9.7 gives a summary.

9.1 MAPs as traffic descriptors in QNs

In this section we introduce MAPs as traffic descriptors in queueing networks. The superposition of two MAPs is explained in Section 9.1.1. In Section 9.1.2 we describe the splitting of a MAP. The analysis of MAP|MAP|1 and MAP|MAP|1|K queues is discussed in Section 9.1.3 and Section 9.1.4, respectively. Finally, the state space explosion problem is briefly discussed in Section 9.1.5.

9.1.1 Superposition of traffic streams

The superposition of two MAPs $(\mathbf{A_0}, \mathbf{A_1})$ and $(\mathbf{B_0}, \mathbf{B_1})$ is a new MAP $(\mathbf{C_0}, \mathbf{C_1})$ with

$$\mathbf{C_0} = \mathbf{A_0} \oplus \mathbf{B_0}, \quad \mathbf{C_1} = \mathbf{A_1} \oplus \mathbf{B_1},$$

where $\mathbf{L} \oplus \mathbf{M} = \mathbf{L} \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{M}$, and $\otimes$ is the Kronecker product operator. If $\boldsymbol{\pi}_\mathbf{A}$ resp. $\boldsymbol{\pi}_\mathbf{B}$ is the steady-state probability vector of the MAP $(\mathbf{A_0}, \mathbf{A_1})$ resp. $(\mathbf{B_0}, \mathbf{B_1})$, then the steady-state probability vector of the resulting MAP is given by $\boldsymbol{\pi}_\mathbf{A} \otimes \boldsymbol{\pi}_\mathbf{B}$.

9.1.2 Splitting traffic streams

The Markovian splitting of a MAP $(\mathbf{A_0}, \mathbf{A_1})$ with probability r gives two MAPs $(\mathbf{B_0}, \mathbf{B_1})$ and $(\mathbf{C_0}, \mathbf{C_1})$ with

$$\begin{aligned}(\mathbf{B_0}, \mathbf{B_1}) &= (\mathbf{A_0} + (1-r)\mathbf{A_1}, r\mathbf{A_1}),\\ (\mathbf{C_0}, \mathbf{C_1}) &= (\mathbf{A_0} + r\mathbf{A_1}, (1-r)\mathbf{A_1}).\end{aligned}$$

The two resulting MAPs have the same steady-state probabilities as the original MAP.

9.1.3 Analysis of MAP|MAP|1 queues

The underlying CTMC

In a MAP|MAP|1 queue both the arrival process and the service process are MAPs. The underlying infinite Markov chain of such a queue is a QBD, where each level of the QBD state space corresponds to a specific number of customers in the queueing system.

Let $(\mathbf{\Lambda_0}, \mathbf{\Lambda_1})$ be the arrival MAP with l states, and $(\mathbf{S_0}, \mathbf{S_1})$ the service MAP with m states. Then the underlying QBD has the following generator matrix

$$\mathbf{Q} = \begin{pmatrix} \mathbf{B_0} & \mathbf{A_0} & 0 & \\ \mathbf{B_1} & \mathbf{A_1} & \mathbf{A_0} & \\ \mathbf{0} & \mathbf{A_2} & \mathbf{A_1} & \\ & \ddots & \ddots & \ddots \end{pmatrix}, \tag{9.1}$$

with

$$\begin{aligned} \mathbf{B_0} &= \mathbf{\Lambda_0} \otimes \mathbf{I}, \\ \mathbf{B_1} &= \mathbf{I} \otimes \mathbf{S_1}, \\ \mathbf{A_0} &= \mathbf{\Lambda_1} \otimes \mathbf{I}, \\ \mathbf{A_1} &= \mathbf{\Lambda_0} \oplus \mathbf{S_0}, \\ \mathbf{A_2} &= \mathbf{I} \otimes \mathbf{S_1}. \end{aligned}$$

Its steady-state probability vector $\boldsymbol{\pi}$ has the following structure

$$\boldsymbol{\pi} = \begin{pmatrix} \mathbf{v_0} & \mathbf{v_1} & \dots \end{pmatrix},$$

where each $\mathbf{v_i}$ is a vector of size $l \cdot m$ and contains the steady-state probabilities for the states of level i in the QBD. In the following, we assume that matrix-geometric solution methods, like the LR-approach (see Section 3.3), are used to compute $\boldsymbol{\pi}$. Matrix-geometric solution methods are based on the fact that the vectors $\mathbf{v_i}$ have the so-called matrix-geometric form

$$\mathbf{v_i} = \mathbf{v_0}\mathbf{R}^i, \quad \mathbf{R} \in \mathbb{R}^{lm \times lm}, \quad i = 0, 1, \dots,$$

where $\mathbf{R}$ is the entry-wise smallest non-negative solution of the matrix-quadratic equation

$$\mathbf{A_0} + \mathbf{R}\mathbf{A_1} + \mathbf{R}^2\mathbf{A_2} = \mathbf{0}.$$

The departure process

In the case of MAP|MAP|1 queues, the output process is an infinite MAP $(\mathbf{D_0}, \mathbf{D_1})$ which can easily be obtained by modifying the underlying Markov chain of the queue.

Since each service completion corresponds to an "arrival" in the departure process we obtain $\mathbf{D_0}$ and $\mathbf{D_1}$ directly from the generator (9.1) by marking the transitions of the sub-matrices $\mathbf{B_1}$ and $\mathbf{A_2}$ as "arrival" transitions, as follows:

$$\mathbf{D_0} = \begin{pmatrix} \mathbf{B_0} & \mathbf{A_0} & \mathbf{0} & & \\ \mathbf{0} & \mathbf{A_1} & \mathbf{A_0} & & \\ & \mathbf{0} & \mathbf{A_1} & & \\ & & \ddots & \ddots & \end{pmatrix}, \quad \mathbf{D_1} = \begin{pmatrix} \mathbf{0} & & & & \\ \mathbf{B_1} & \mathbf{0} & & & \\ \mathbf{0} & \mathbf{A_2} & \mathbf{0} & & \\ & & \ddots & \ddots & \end{pmatrix}.$$

Node performance

Having obtained the probability vector $\boldsymbol{\pi}$, many performance measures of the queue can be computed easily, e.g., the moments of the queue length distribution:

$$\mathrm{E}[N^k] = \sum_{i=0}^{\infty} i^k \mathbf{v_i 1} = \sum_{i=0}^{\infty} i^k \mathbf{v_0} \mathbf{R}^i \mathbf{1},$$

which yields, in case $k = 1$:

$$\mathrm{E}[N] = \mathbf{v_0 R}(\mathbf{I} - \mathbf{R})^{-2}\mathbf{1}.$$

9.1.4 Analysis of MAP|MAP|1|K queues

The underlying CTMC

Naturally, the underlying CTMC of a MAP|MAP|1|K queue corresponds to a finite QBD. Let $(\mathbf{\Lambda_0}, \mathbf{\Lambda_1})$ be the arrival MAP with l states and $(\mathbf{S_0}, \mathbf{S_1})$ the service MAP with m states. Then the underlying QBD has the following generator matrix

$$\mathbf{Q} = \begin{pmatrix} \mathbf{B_0} & \mathbf{A_0} & \mathbf{0} & & & \\ \mathbf{B_1} & \mathbf{A_1} & \mathbf{A_0} & & & \\ \mathbf{0} & \mathbf{A_2} & \mathbf{A_1} & & & \\ & & \ddots & \ddots & \ddots & \\ & & & \mathbf{A_2} & \mathbf{A_1} & \mathbf{A_0} \\ & & & & \mathbf{A_2} & \mathbf{C} \end{pmatrix}, \tag{9.2}$$

with

$$\begin{aligned} \mathbf{B_0} &= \mathbf{\Lambda_0} \otimes \mathbf{I}, \\ \mathbf{B_1} &= \mathbf{I} \otimes \mathbf{S_1}, \\ \mathbf{A_0} &= \mathbf{\Lambda_1} \otimes \mathbf{I}, \\ \mathbf{A_1} &= \mathbf{\Lambda_0} \oplus \mathbf{S_0}, \\ \mathbf{A_2} &= \mathbf{I} \otimes \mathbf{S_1}, \\ \mathbf{C} &= (\mathbf{\Lambda_0} + \mathbf{\Lambda_1}) \oplus \mathbf{S_0}. \end{aligned}$$

Consequently, its steady-state probability vector $\boldsymbol{\pi}$ is finite, too, and has a structure similar to the infinite case:

$$\boldsymbol{\pi} = \begin{pmatrix} \mathbf{v_0} & \mathbf{v_1} & \dots & \mathbf{v_K} \end{pmatrix},$$

where each $\mathbf{v_i}$ is a vector of size $l \cdot m$ and contains the steady-state probabilities for the states of level i in the QBD.

The departure process

In analogy to the case of MAP|MAP|1 queues we give the output process of the MAP|MAP|1|K as a finite MAP $(\mathbf{D_0}, \mathbf{D_1})$ with

$$\mathbf{D_0} = \begin{pmatrix} \mathbf{B_0} & \mathbf{A_0} & \mathbf{0} & & & & \\ \mathbf{0} & \mathbf{A_1} & \mathbf{A_0} & & & & \\ & \mathbf{0} & \mathbf{A_1} & & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & & \mathbf{0} & \mathbf{A_1} & \mathbf{A_0} \\ & & & & & \mathbf{0} & \mathbf{C} \end{pmatrix}, \quad \mathbf{D_1} = \begin{pmatrix} \mathbf{0} & & & & & & \\ \mathbf{B_1} & \mathbf{0} & & & & & \\ \mathbf{0} & \mathbf{A_2} & \mathbf{0} & & & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & \mathbf{A_2} & \mathbf{0} & \mathbf{0} \\ & & & & & \mathbf{A_2} & \mathbf{0} \end{pmatrix}.$$

Node performance

Since the probability vector $\boldsymbol{\pi}$ is finite, performance measures like the mean queue length can easily be calculated.

9.1.5 State space size

A network analysis relying solely on MAPs as traffic descriptors suffers under the state space explosion problem. Given two MAPs of size m resp. n, the superposition of them is a MAP of size $m \cdot n$. Splitting a MAP of size m will result in two MAPs of the same size. The service operation at a queue with finite capacity K will generate a departure MAP whose size is the product of the capacity K, the size of the arrival MAP and the size of the service MAP. Correspondingly, the departure MAP of queues with infinite size is an infinite MAP. Obviously, when we apply our decomposition based analysis method (see Section 2.3) to a realistic network (i.e., non-exponential service processes and queueing capacities larger than 1) we quickly obtain MAPs that are far too large for the numerical treatment in a reasonable time. Hence, reduction methods are required that should be applied during the analysis of the network whenever a MAP's size exceeds a given threshold. Such methods are discussed in the following sections.

9.2 Reduction by removing equivalent states

The idea of this technique is to remove states of the MAP that are equivalent (see below for a formal definition of state equivalence). If two states a and b of a MAP are equivalent we can remove one of them, e.g., a, without changing the behavior of the MAP.

We first introduce the notion of equivalence relations on states for CTMCs in Section 9.2.1 and describe an algorithm to compute the equivalence relation. Then we explain the required modifications in order to extend the algorithm to MAPs in Section 9.2.2.

9.2.1 Equivalence for CTMCs

Let $\mathbf{Q}$ be the generator matrix of a CTMC with state space S. Let γ_Q be the function that calculates the cumulative rate to reach a set of states $C \subset S$ from a single state r:

$$\gamma_Q(r, C) = \sum_{c \in C} Q_{rc}$$

Two states $i, j \in S$ are called *equivalent* iff

- $i = j$, or
- $i \neq j$ and $\gamma_Q(i, C) = \gamma_Q(j, C)$ for all equivalence classes C.

This notion of equivalence is the strong Markovian bisimulation equivalence [50, 51] and coincides with the notion of lumpability. The equivalence classes of states for a CTMC with generator matrix $\mathbf{Q}$ and state space S can be computed with the algorithm presented in [51]. It is shown in Figure 9.1. The algorithm divides the state space of the CTMC into partitions that are successively refined until each partition only contains equivalent states. During refinement, the partitions are split into smaller partitions with respect to so-called splitters. We begin with one large partition containing all states (line 2) which also acts as splitter class (line 3). As long as splitters are available (line 4; each splitter is only used once), one splitter is chosen (line 5) for the refinement of all partitions (line 7). The new partitions generated by the refinement are added to the set of splitters and the used splitter is removed (line 8 and 9). The function refine_Q refines a partition P by means of a splitter *Spl*. It forms new partitions containing the states that have the same cumulative rate to the splitter:

$$\mathit{refine}_Q(P, \mathit{Spl}) = \bigcup_{v \in \mathbb{R}^+} \{\{s \in P | \gamma_Q(s, \mathit{Spl}) = v\}\} .$$

[51] also describes an efficient implementation of the algorithm where the states are hold in a tree, sorted by their cumulative rates to the splitters. The algorithm has

```
procedure $computeEqClasses(S, \mathbf{Q})$
    $Partitions := \{S\}$
    $Splitters := \{S\}$
    while $Splitters \neq \emptyset$ do
        choose $Spl \in Splitters$
        $OldParts := Partitions$
        $Partitions := \bigcup_{P \in Partitions} refine_Q(P, Spl) - \{\emptyset\}$
        $NewParts := Partitions - OldParts$
        $Splitters := (Splitters - \{Spl\}) \cup NewParts$
    end
    return $Partitions$
end
```

Figure 9.1: Algorithm for computing equivalence classes of states

a time complexity of $O(m \log n)$ and a space complexity of $O(m + n)$, where n is the number of states and m is the number of transitions. Note that the algorithm is much less computation intensive than reduction algorithms that explore the CTMC via its steady-state probabilities.

9.2.2 Approximate equivalence for MAPs

If we want to apply the algorithm to a MAP $(\mathbf{D_0}, \mathbf{D_1})$ we have to separately handle the transitions specified in $\mathbf{D_0}$ and $\mathbf{D_1}$. Instead of $\gamma_Q(r, C)$ we now have

$$\gamma_0(r, C) = \sum_{c \in C} (\mathbf{D_0})_{rc} \quad \text{and} \quad \gamma_1(r, C) = \sum_{c \in C} (\mathbf{D_1})_{rc}.$$

However, when using MAPs as traffic descriptors, the condition

$$\gamma_0(i, C) = \gamma_0(j, C) \wedge \gamma_1(i, C) = \gamma_1(j, C)$$

is rarely fulfilled for two states i, j. Even if the MAP actually has equivalent states, computational inaccuracies may prevent to identify them. Hence, we introduce the notion of “approximate” bisimulation where two states $i \neq j$ are equivalent iff

$$|\gamma_0(i, C) - \gamma_0(j, C)| < \varepsilon_0 \quad \wedge \quad |\gamma_1(i, C) - \gamma_1(j, C)| < \varepsilon_1$$

for all equivalence classes C. By choosing ε_0 and ε_1 we can control the quality of the approximation.

Knowing the approximate equivalence classes, we have to be more cautious in the construction of the reduced MAP than in the case of the exact equivalence. Let C and D be two equivalence classes. In the case of exact equivalence, the rate of the transition from C to D is $\gamma_Q(c, D)$, where c is an arbitrary (representative) state of C. If we use approximate equivalence, the states in C have different cumulative rates to D. In order to reduce errors in the approximation, we define

$$\frac{1}{|C|} \sum_{c \in C} \gamma_Q(c, D)$$

as the (average) transition rate to be used.

9.3 Finite output process approximation for MAP|MAP|1 queues

In [9], Bean *et al.* presented an approach to approximate the departure process of MAP|PH|1 queues by finite MAPs. This approach can be easily extended[1] to MAP|MAP|1 queues [102]. The thus obtained MAP will have two important characteristics:

1. It *exactly* describes the inter-departure time distribution of the original departure process [9, 49].
2. It is able to describe the correlation structure of the original departure process *at arbitrary precision* albeit at the cost of, possibly, a large state space.

The second characteristic implies that, depending on the required quality of the approximation, the obtained MAP may become so large that other reduction methods must be applied afterwards.

The approximation is based on the structural equality of the infinite departure MAP and the underlying CTMC of the QBD describing the MAP|MAP|1 queue: each level in the QBD has its corresponding entries in the matrices of the departure MAP (see Section 9.1.3). However, it often is not required to represent all levels of the QBD in the departure process. Let $\boldsymbol{\pi} = (\ \mathbf{v_0} \ \ \mathbf{v_1} \ \ \ldots\ )$ be the steady-state probability vector of the CTMC underlying the QBD. Since the probability $\mathbf{v_i e}$ to be in level i decreases with increasing i, there is a level s where $\mathbf{v_s e}$ becomes so small that one may decide not to represent the levels s and higher in full detail in the departure process.

In the following we assume $s > 1$, i.e., level s is not adjacent to the border level 0. We transform the infinite (exact) departure process into a finite (approximated) departure process by condensing the levels s and higher to only *one* level, the so-called called *clipping level* $\widehat{s}$. The states of the level $\widehat{s}$ in the approximate MAP will

[1]Actually, we developed our approach while not being aware of [9].

have the steady-state probability $\mathbf{v_{\widehat{s}}}$ with

$$\mathbf{v_{\widehat{s}}} = \sum_{i \geq s} \mathbf{v_i} = \sum_{i=s}^{\infty} \mathbf{v_0} \mathbf{R}^i = \mathbf{v_0} \mathbf{R}^s (\mathbf{I} - \mathbf{R})^{-1}. \tag{9.3}$$

The transition rates of the approximate MAP are computed as follows:

- The levels $\{0, \ldots, \widehat{s} - 1\}$ in the approximate MAP have the same transition rates to their neighbor levels and to the same level as the corresponding levels $\{0, \ldots, s - 1\}$ of the original MAP.
- In the original MAP, a transition from the level $t \geq s$ to the same level t is described by the block matrix $\mathbf{A_1}$ and a transition from t to the higher level $t + 1$ is described by the matrix $\mathbf{A_0}$. In the approximate MAP, these levels are all condensed to the level $\widehat{s}$. Hence $\mathbf{A_0} + \mathbf{A_1}$ describes the (not marked) transitions from level $\widehat{s}$ to itself.
- In the original MAP, events are generated by a transition from a level t to the lower level $t - 1$, described by the matrix $\mathbf{A_2}$ for $t > 1$, respectively, $\mathbf{B_1}$ for $t = 1$. If $t > s$, both levels t and $t - 1$ will be collapsed into the level $\widehat{s}$ in the approximate MAP and the transition between them will become a transition from $\widehat{s}$ to $\widehat{s}$. This transition is described by the matrix $\mathbf{A_{2,stay}}$ with

  $$(\mathbf{A_{2,stay}})_{ij} = \frac{1}{v_{\widehat{s},i}} (v_{\widehat{s},i} - v_{s,i}) A_{2,ij}.$$

  If $t = s$ in the original MAP, only level t will be collapsed into $\widehat{s}$ in the approximate MAP and the transition between t and $t - 1$ becomes a transition from $\widehat{s}$ to $\widehat{s} - 1$, described by the matrix $\mathbf{A_{2,down}}$ with

  $$(\mathbf{A_{2,down}})_{ij} = \frac{1}{v_{\widehat{s},i}} v_{s,i} A_{2,ij}.$$

In total, the finite approximate MAP $(\widehat{\mathbf{D}}_\mathbf{0}, \widehat{\mathbf{D}}_\mathbf{1})$ is defined by

$$\widehat{\mathbf{D}}_\mathbf{0} = \begin{pmatrix} \mathbf{B_0} & \mathbf{A_0} & & & \\ \mathbf{0} & \mathbf{A_1} & \mathbf{A_0} & & \\ & \mathbf{0} & \mathbf{A_1} & & \\ & \ddots & \ddots & \ddots & \\ & & & \mathbf{A_0} & \\ & & & \mathbf{A_0} + \mathbf{A_1} & \end{pmatrix}, \text{ and}$$

$$\widehat{\mathbf{D}}_\mathbf{1} = \begin{pmatrix} \mathbf{0} & & & & \\ \mathbf{B_1} & \mathbf{0} & & & \\ & \mathbf{A_2} & 0 & & \\ & & \ddots & \ddots & \\ & & & \mathbf{0} & \\ & & & \mathbf{A_{2,down}} & \mathbf{A_{2,stay}} \end{pmatrix},$$

with

$$\begin{aligned}(\mathbf{A_{2,stay}})_{ij} &= \frac{1}{v_{\widehat{s},i}}(v_{\widehat{s},i} - v_{s,i})A_{2,ij},\\ (\mathbf{A_{2,down}})_{ij} &= \frac{1}{v_{\widehat{s},i}}v_{s,i}A_{2,ij}.\end{aligned}$$

Green proved for MAP|PH|1 queues in [40] that this s-level approximation captures the first $s-1$ correlation coefficients of the departure process exactly. His proof is general enough to be applicable to MAP|MAP|1 queues, as well.

Note that an efficient approach [49] exists to perform the approximation which avoids the computation of $\mathbf{v_{\widehat{s}}}$ (Equation (9.3)), provided that the queue length distribution is not desired.

9.4 Approximation for MAP|MAP|1(|K) queues

Similar to the previous method, this approach tries to condense a set of levels into one level $\widehat{s}$. This time, we aim to merge a finite set of levels $\{s, s+1, \ldots, t\}$ with $1 < s < t$ (and $t < K$ in case of a MAP|MAP|1|K queue). If $\boldsymbol{\pi} = (\ \mathbf{v_0}\ \ \mathbf{v_1}\ \ \ldots\ )$ is the steady-state probability vector of the original departure process, the approximate MAP will have the steady-state probability vector $\widehat{\boldsymbol{\pi}}$ with

$$\widehat{\boldsymbol{\pi}} = (\ \mathbf{v_0}\ \ \mathbf{v_1}\ \ \ldots\ \ \mathbf{v_{s-1}}\ \ \mathbf{v_{\widehat{s}}}\ \ \mathbf{v_{t+1}}\ \ \ldots\ ),$$

where

$$\mathbf{v_{\widehat{s}}} = \sum_{i=s}^{t} \mathbf{v_i}.$$

As in the previous method, the levels not condensed in the approximate MAP have the same transition rates to their neighbor levels and to the same level as the corresponding levels of the original MAP. And again, the event-generating transitions from the level $\widehat{s}$ to the same level and to level $\widehat{s}-1$ are described by

$$\begin{aligned}(\mathbf{A_{2,stay}})_{ij} &= \frac{1}{v_{\widehat{s},i}}(v_{\widehat{s},i} - v_{s,i})A_{2,ij},\\ (\mathbf{A_{2,down}})_{ij} &= \frac{1}{v_{\widehat{s},i}}v_{s,i}A_{2,ij}.\end{aligned}$$

This time we additionally have to consider transitions from $\widehat{s}$ to the higher level $\widehat{s}+1$. These non-marked transitions are specified in analogy to $\mathbf{A_{2,stay}}$ and $\mathbf{A_{2,down}}$ by the matrices $\mathbf{A_{0,stay}}$ and $\mathbf{A_{0,up}}$ with

$$\begin{aligned}(\mathbf{A_{0,stay}})_{ij} &= \frac{1}{v_{\widehat{s},i}}(v_{\widehat{s},i} - v_{t,i})A_{0,ij},\\ (\mathbf{A_{0,up}})_{ij} &= \frac{1}{v_{\widehat{s},i}}v_{t,i}A_{0,ij}.\end{aligned}$$

The finite approximate MAP $(\widehat{\mathbf{D}}_\mathbf{0}, \widehat{\mathbf{D}}_\mathbf{1})$ thus is defined by

$$\widehat{\mathbf{D}}_\mathbf{0} = \begin{pmatrix} \mathbf{B_0} & \mathbf{A_0} & & & & \\ \mathbf{0} & \mathbf{A_1} & \mathbf{A_0} & & & \\ & 0 & \ddots & & & \\ & & & \mathbf{A_0} & & \\ & & & \mathbf{A_1} + \mathbf{A_{0,stay}} & \mathbf{A_{0,up}} & \\ & & & 0 & \mathbf{A_1} & \\ & & & & 0 & \ddots \end{pmatrix},$$

$$\widehat{\mathbf{D}}_\mathbf{1} = \begin{pmatrix} 0 & & & & \\ \mathbf{B_1} & 0 & & & \\ & \mathbf{A_2} & \ddots & & \\ & & & 0 & \\ & & & \mathbf{A_{2,down}} & \mathbf{A_{2,stay}} & \\ & & & & \mathbf{A_2} & \ddots \end{pmatrix},$$

with

$$\begin{aligned} (\mathbf{A_{0,stay}})_{ij} &= \frac{1}{v_{\widehat{s},i}}(v_{\widehat{s},i} - v_{t,i})A_{0,ij}, \\ (\mathbf{A_{0,up}})_{ij} &= \frac{1}{v_{\widehat{s},i}}v_{t,i}A_{0,ij}, \\ (\mathbf{A_{2,stay}})_{ij} &= \frac{1}{v_{\widehat{s},i}}(v_{\widehat{s},i} - v_{s,i})A_{2,ij}, \\ (\mathbf{A_{2,down}})_{ij} &= \frac{1}{v_{\widehat{s},i}}v_{s,i}A_{2,ij}. \end{aligned}$$

One can easily verify that the stationary inter-departure time distribution of the original departure MAP is preserved.

9.5 Approximation of processes with positive, exponentially decreasing autocovariance

In the following we describe how a MAP can be constructed that fits a given first, second, and third moment and a given positive, exponentially decreasing autocovariance function.

The aimed-at MAP

Consider the PH distributions $(\boldsymbol{\alpha}, \mathbf{A})$ and $(\boldsymbol{\beta}, \mathbf{B})$ with inter-event time X_A resp. X_B. Given two probabilities p and q, the two distributions can be combined to form a MAP $(\mathbf{D_0}, \mathbf{D_1})$ in the following way:

$$\mathbf{D_0} = \begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} \end{pmatrix}, \quad \mathbf{D_1} = \begin{pmatrix} p\mathbf{A^0}\boldsymbol{\alpha} & (1-p)\mathbf{A^0}\boldsymbol{\beta} \\ (1-q)\mathbf{B^0}\boldsymbol{\alpha} & q\mathbf{B^0}\boldsymbol{\beta} \end{pmatrix},$$

i.e., the MAP switches with probability $1-p$ to $(\boldsymbol{\beta}, \mathbf{B})$ after an event generated by $(\boldsymbol{\alpha}, \mathbf{A})$ and vice versa with probability $1-q$. The first and second moment of the inter-event time X_D of the MAP are given by

$$\begin{aligned} \mathrm{E}[X_D] &= x\mathrm{E}[X_A] + y\mathrm{E}[X_B], \\ \mathrm{E}[X_D^2] &= x\mathrm{E}[X_A^2] + y\mathrm{E}[X_B^2], \\ \mathrm{E}[X_D^3] &= x\mathrm{E}[X_A^3] + y\mathrm{E}[X_B^3] \end{aligned}$$

with $x = \frac{1-q}{2-p-q}$ and $y = \frac{1-p}{2-p-q}$. The autocovariance $R(1)$ for time lag 1 is given by

$$\begin{aligned} R(1) &= \mathrm{E}[X_{D,1}X_{D,2}] - \mathrm{E}[X_D]^2 \\ &= x\mathrm{E}[X_A]\left(p\mathrm{E}[X_A] + (1-p)\mathrm{E}[X_B]\right) \\ &\quad + y\mathrm{E}[X_B]\left(q\mathrm{E}[X_B] + (1-q)\mathrm{E}[X_A]\right) - \mathrm{E}[X_D]^2. \end{aligned}$$

If we choose $p = q$ the expression simplifies to

$$R(1) = \frac{1}{4}(2p-1)(\mathrm{E}[X_A] - \mathrm{E}[X_B])^2,$$

or, more generally, for a time lag $k > 0$:

$$R(k) = \frac{1}{4}(2p-1)^k(n-1)^2\mathrm{E}[X_A]^2 \tag{9.4}$$

where $n = \mathrm{E}[X_B]/\mathrm{E}[X_A]$. Additionally, the moments become

$$\mathrm{E}[X_D] = \frac{n+1}{2}\mathrm{E}[X_A], \tag{9.5}$$

$$\mathrm{E}[X_D^2] = \frac{1}{2}\mathrm{E}[X_A^2] + \frac{1}{2}\mathrm{E}[X_B^2], \tag{9.6}$$

$$\mathrm{E}[X_D^3] = \frac{1}{2}\mathrm{E}[X_A^3] + \frac{1}{2}\mathrm{E}[X_B^3]. \tag{9.7}$$

The fitting procedure

In a fitting problem, the three moment $\mathrm{E}[X_D]$, $\mathrm{E}[X_D^2]$, $\mathrm{E}[X_D^3]$ and the autocovariance function R are given and we want to find $(\boldsymbol{\alpha}, \mathbf{A})$, $(\boldsymbol{\beta}, \mathbf{B})$, p and q in order to construct the MAP $(\mathbf{D_0}, \mathbf{D_1})$. We can compute them using the following procedure:

1. We choose $p = q$. First, we determine the base $2p-1$ (and hence, p and q) of the exponential fall-off of $R(k)$. Following Equation (9.4), we have $2p-1 = \frac{R(k+1)}{R(k)}$ for any $k > 0$. Of course, this is only true if R really is an exponentially decreasing function. In practice, this is usually not the case and we have to approximate the shape of R, for example by choosing $p := (\frac{R(2)}{R(1)} + 1)/2$.

2. Equation (9.5) yields $\mathrm{E}[X_A] = 2\mathrm{E}[X_D]/(n+1)$. By substituting $\mathrm{E}[X_A]$ in Equation (9.4) and solving to n, we obtain
$$n := (\mathrm{E}[X_D] \pm \sqrt{r})^2/(\mathrm{E}[X_D]^2 - r),$$
where $r = \frac{R(k)}{(2p-1)^k}$ for any $k > 0$, provided that $R(k)$ is an exponentially decreasing function. Otherwise, only an approximation to $R(k)$ is obtained. For example, if we choose $r := \frac{R(t)}{(2p-1)^t}$ for a fixed $t > 0$, the resulting MAP will have the autocovariance function R_{fit} with $R_{fit}(t) = R(t)$.
3. Following Equations (9.6) and (9.7), we have some freedom for the second and third moments of the involved PH distributions. We choose $\mathrm{E}[X_A^2] := \mathrm{E}[X_B^2] := \mathrm{E}[X_D^2]$ and $\mathrm{E}[X_A^3] := \mathrm{E}[X_B^3] := \mathrm{E}[X_D^3]$.
4. Finally, we have to find the PH distributions $(\boldsymbol{\alpha}, \mathbf{A})$ and $(\boldsymbol{\beta}, \mathbf{B})$ fitting $\mathrm{E}[X_A]$, $\mathrm{E}[X_A^2]$, $\mathrm{E}[X_A^3]$, respectively $\mathrm{E}[X_B]$, $\mathrm{E}[X_B^2]$, $\mathrm{E}[X_B^3]$. We fit Minimal Acyclic PH distributions, as described in Section 5.5.5.

This method can be considered as a representant of a class of very similar fitting techniques where PH distributions are combined in various ways to create MAPs with specific distributions and autocovariance functions. For example, modifications in the arrangement or number of the involved PH distributions result in autocovariance functions with negative values, et cetera.

9.6 Fitting PH renewal processes to MAPs

The most obvious method to reduce the size of a MAP is to ignore its higher-order statistics. Although higher-order statistics of arrival processes (e.g., autocorrelation) heavily influence the performance of queueing stations it is clear that, in general, the most important aspect of an arrival process is its inter-arrival time distribution.

The inter-event time distribution of a MAP can be naturally extracted by reducing the MAP to a PH renewal process: Given a MAP $(\mathbf{Q_0}, \mathbf{Q_1})$ with steady-state probability vector $\boldsymbol{\pi}$ and initial probability vector $\mathbf{p} = \frac{1}{\boldsymbol{\pi}\mathbf{Q_1}\mathbf{1}}\boldsymbol{\pi}\mathbf{Q_1}$, we can derive the PH renewal process $(\mathbf{p}, \mathbf{Q_0})$ with identical inter-event time distribution (see Section 3.2). Note that for an arbitrary MAP $(\mathbf{Q_0}, \mathbf{Q_1})$ it does not really make sense to take $(\mathbf{p}, \mathbf{Q_0})$ instead of $(\mathbf{Q_0}, \mathbf{Q_1})$ since both representations have the same number of states. To achieve a state space reduction one would have to apply a moment-fitting algorithm afterwards as described in Section 5.5.5.

However, if the MAP is the infinite output process of a MAP|MAP|1 queue one can make use of its regular structure to find a compact PH representation [88]. Using the notation introduced in Section 9.1.3 the vector $\mathbf{p}$ becomes
$$\mathbf{p} = \begin{pmatrix} \mathbf{p_0} & \mathbf{p_1} & \cdots \end{pmatrix},$$

where

$$\begin{aligned}\mathbf{p_0} &= c\mathbf{v_0}\mathbf{R}\mathbf{B_1},\\ \mathbf{p_i} &= c\mathbf{v_0}\mathbf{R}^{i+1}\mathbf{A_2},\\ \text{with } c &= \frac{1}{\mathbf{v_0}\mathbf{R}(\mathbf{I}-\mathbf{R})^{-1}\mathbf{A_2}\mathbf{1}}.\end{aligned}$$

By following the approach described in Section 9.3 we merge all levels > 0 to one single level and obtain a PH renewal process $(\mathbf{p'},\mathbf{Q'})$ with identical inter-event time distribution where

$$\begin{aligned}\mathbf{p'} &= \begin{pmatrix}\mathbf{p_0} & \mathbf{p'_1}\end{pmatrix},\\ \mathbf{Q'} &= \begin{pmatrix}\mathbf{B_0} & \mathbf{A_0}\\ \mathbf{0} & \mathbf{A_0}+\mathbf{A_1}\end{pmatrix},\end{aligned}$$

with

$$\begin{aligned}\mathbf{p'_1} &= \sum_{i=1}^{\infty}\mathbf{p_i}\\ &= c\mathbf{v_0}\mathbf{R}^2(\mathbf{I}-\mathbf{R})^{-1}\mathbf{A_2}.\end{aligned}$$

9.7 Summary and conclusions

In this chapter, we have introduced MAPs as traffic descriptors in the analysis of queueing networks. However, the approach suffers under the state space explosion problem since the service operation and the traffic splitting operation increase the size of the descriptors in each iteration. Hence, we have presented five different MAP reduction methods in order to remove or attenuate the effect of the state space explosion. The first method tries to reduce the size of a MAP by identifying states that are approximately equivalent. Unlike other methods, this does not require the computation of the steady-state probabilities of the MAP. The next two methods approximate the departure process of MAP|MAP|1(|K) queues. The resulting MAPs have the same inter-departure time distribution as the original departure process and approximate correlations of the inter-departure time. The fourth method fits a MAP with a positive, decreasing autocovariance function to a point process. It preserves the first, second and third moment of the inter-event time and approximates the autocovariance function by an exponential function. This method is the most computational intensive because it does not only require the moments of the process but also single values of the autocovariance function in order to determine its parameters. The fifth method approximates the infinite departure MAP of a MAP|MAP|1 queue by a finite PH renewal process with the same inter-event time distribution. Naturally, correlations in the inter-event time are ignored. The performance of all these methods is discussed in the next chapter.

Chapter 10

Performance of MAP-Based Traffic Descriptors

In this chapter, we study the performance of the MAP-based traffic descriptors in combination with the approximation and reduction techniques presented in the previous chapter.

We begin with the reduction method based on removing equivalent states in Section 10.1. In Section 10.2, we approximate the departure process of MAP|MAP|1 queues by using the methods from Section 9.3 and 9.4. The approximation method from Section 9.4 is also used for the analysis of MAP|MAP|1|K queues in Section 10.3. The approximation of processes with positive, exponentially decreasing autocovariance is tested in Section 10.4. The approach to fit PH renewal processes to MAPs (see Section 9.6) is not separately tested but applications of the method can be found in Sections 10.2, 10.3 and 10.4. We conclude this chapter with Section 10.5.

The results of the numerical analysis are compared to results determined using discrete-event simulation, or, where available, to exact analytical solutions. If not stated otherwise, arrival time and service time distributions specified by their rate and the squared coefficient of variation (SCV) are always mapped to the PH distributions of the original FiFiQueues algorithm. Relative errors between numerical analysis and simulation/exact solution are always computed relative to the latter.

10.1 Reduction by removing equivalent states

Although attractive in theory, our experiments have shown that the MAPs generated during the queueing network analysis generally do not have equivalent states that can removed by the algorithm described in Section 9.2, even if large error bounds ε_1 and ε_2 are chosen. This is because the involved operations like splitting, superposition, and queueing service generate very regularly structured MAPs where each state has its unique meaning.

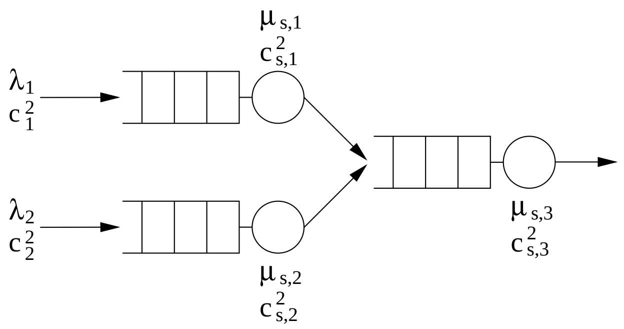


Figure 10.1: Queueing network with the superposition of two similar traffic descriptors

Nevertheless, the algorithm can detect equivalences in a MAP that has been created by the superposition of two nearly identical MAPs. To test this, we evaluate the queueing network shown in Figure 10.1. We feed two hyper-exponential PH renewal processes with $\lambda_1 = 2, c_1^2 = 4$, resp. $\lambda_2 = 2.1, c_2^2 = 4$ into two identical queueing stations with infinite queueing capacity and hyper-exponential service time distribution ($\mu_{s,1} = \mu_{s,2} = 3$ and $c_{s,1}^2 = c_{s,2}^2 = 2$). We apply the clipping algorithm of Section 9.3 to the departure processes of the two queues in order to obtain two MAPs with 16 states. Note that the two MAPs do not only differ in the departure rate but also in the steady-state probabilities of their states since the two stations experience different loads. We superposition both MAPs and use the resulting MAP as arrival process for a subsequent queueing station with hypo-exponential service time distribution ($\mu_{s,3} = 5, c_{s,3}^2 = 0.5$). The experiment is also done for $\lambda_2 = 2.2$ and $\lambda_2 = 2.4$.

Then we repeat the experiments but this time we remove equivalent states from the superpositioned MAP. We have to increase the error bounds in order to find equivalent states for increasing λ_2. For $\lambda_2 = 2.1$ a relative error bound of $\varepsilon = 5\%$ (for the cumulative rate functions γ_0 and γ_1) is sufficient to remove nearly 50% of the states. For $\lambda_2 = 2.2$, the error bound must be at least 10%, and for $\lambda_2 = 2.4$ we have to set it to 33%. In Table 10.1, we compare the characteristics of the original superpositioned MAP and the reduced MAP, as well as the the mean queue length E[N] of the last queueing station. The rows labeled λ, c^2, γ and I show the arrival rate, the SCV, and the skewness of the inter-arrival time, resp. the limiting index of dispersion for the counting process. The last column gives the errors caused by the reduction, relative to the results of the original MAP. As expected, the errors increase with λ_2 (with dramatic consequences for the mean queue length in case of $\lambda_2 = 2.4$ because of the high load). Similar results are obtained if we vary the service rates or the SCVs instead of the arrival rates. Table 10.2 shows the results for different values of $c_{s,2}^2$ (with $\lambda_1 = \lambda_2 = 2$).

The results show that the method can be successfully applied in case of small

λ_2	measure	original	reduced	rel. error
2.1	#states	256	136	
($\varepsilon = 5\%$)	λ	4.10	4.10	0.0%
	c^2	2.02	2.01	-0.5%
	γ	4.93	4.91	-0.4%
	I	3.39	3.37	-0.6%
	E[N]	7.19	7.22	0.4%
2.2	#states	256	136	
($\varepsilon = 10\%$)	λ	4.20	4.21	0.2%
	c^2	1.99	1.98	-0.5%
	γ	4.87	4.83	-0.8%
	I	3.33	3.30	-0.9%
	E[N]	8.35	8.45	1.8%
2.4	#states	256	141	
($\varepsilon = 33\%$)	λ	4.40	4.68	6.4%
	c^2	1.92	1.71	-10.9%
	γ	4.71	4.12	-12.5%
	I	3.19	2.68	-16.0%
	E[N]	11.78	21.53	82.8%

Table 10.1: Comparison of the results for the original superpositioned MAP and the reduced MAP for different λ_2

$c^2_{s,2}$	measure	original	reduced	rel. error
2.1	#states	256	136	
($\varepsilon = 5\%$)	E[N]	6.29	6.33	0.6%
2.2	#states	256	136	
($\varepsilon = 15\%$)	E[N]	6.33	6.41	1.3%
2.4	#states	256	136	
($\varepsilon = 15\%$)	E[N]	6.42	6.56	2.2%
2.6	#states	256	148	
($\varepsilon = 32\%$)	E[N]	6.50	7.00	7.7%
2.8	#states	256	188	
($\varepsilon = 34\%$)	E[N]	6.58	12.60	91.5%

Table 10.2: Comparison of the results for the original superpositioned MAP and the reduced MAP for different $c^2_{s,2}$

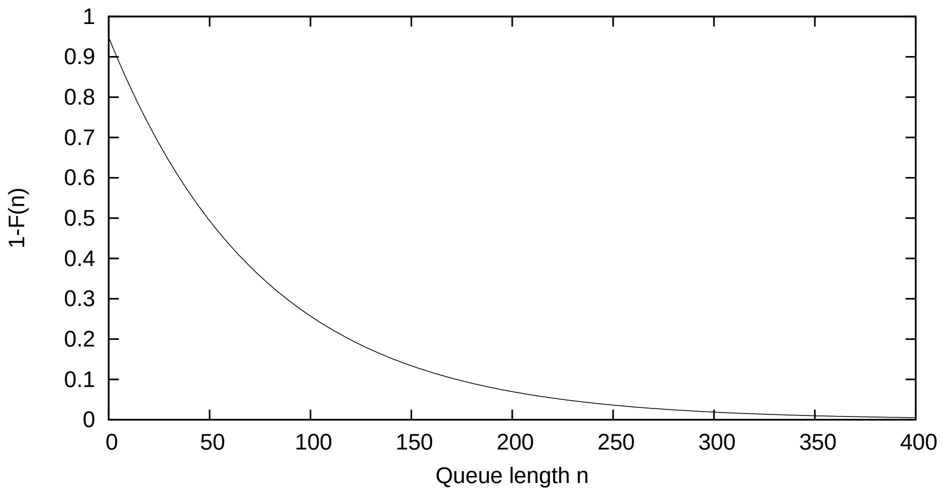


Figure 10.2: Complementary distribution function of the queue length

differences between the superpositioned traffic descriptors. The low time complexity of the algorithm (in comparison to the queueing station analysis) suggests to apply it in a QN analysis after each superposition. In order to ensure the quality of the approximation, the error bounds must be chosen (for example, using a Newton-iteration) such that important characteristics of the reduced MAP, such as rate and SCV, are conserved.

10.2 Approximations for MAP|MAP|1 queues

In this section we evaluate the approximation methods described in Section 9.3 and 9.4 for MAP|MAP|1 queues. We begin with a study of a single queueing station in Section 10.2.1. Then we evaluate a tandem queueing network with various input processes in Section 10.2.2. In Section 10.2.3, we test MAPs in a queueing network with traffic splitting. A summary of the results is given in Section 10.2.4.

10.2.1 Study of a single queueing station

We start our evaluation with the analysis of a simple PH|PH|1 queue using an Erlang-2 service process with service rate 1. The queue is fed by a hyper-exponential renewal process with arrival rate 0.95 and SCV 8. When analyzing this queue we can see that the high load and the high variance of the input process lead to a high mean queue length of nearly 77 with a corresponding slowly decaying queue length distribution function (Figure 10.2).

First, we investigate the departure process approximated by the simple approximation method presented in Section 9.3. How does the quality of the approximation depend on the choice of the clipping level? In view of the slowly decaying queue

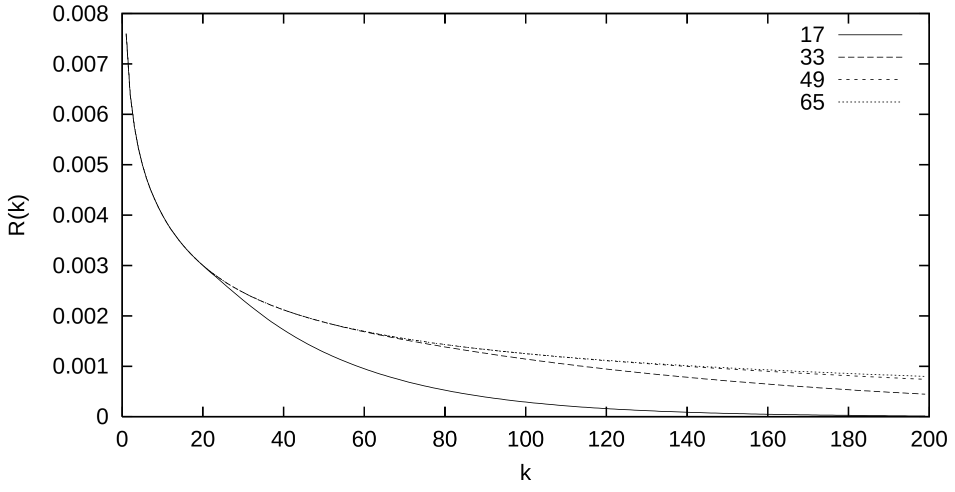


Figure 10.3: Autocovariance function $R(k)$ of the approximate departure MAP for clipping levels 17, 33, 49 and 65

length distribution it is clear that a high clipping level is required to preserve the characteristics of the original departure process. To illustrate this, we have computed the approximated departure process for four different clipping levels, namely 17, 33, 49 and 65. For these cases, the autocovariance functions $R(k)$ are shown in Figure 10.3. Choosing 33 as clipping level seems to result in a good approximation to the autocovariance function of the exact departure process (which has not been shown here because it nearly is equal to the 65-approximation).

These results show that even MAPs with low clipping level can give a good approximation of the correlation structure at small time lags k. In fact, Green proved for MAP|PH|1 queues in [40] that the k-level approximation captures the first $k-1$ correlation coefficients of the departure process exactly, which is confirmed by Figure 10.3. For the approximation of correlation at large time lags, MAPs must be chosen which respect the structure of the higher levels of the queue QBD. However, the figure also shows that clipping levels which are considerably smaller than the mean queue length yield good approximations to the exact departure process. This may be surprising since clipping levels like 65 only catch 55% of the queue length distribution probability (Figure 10.2). But we should note that many important characteristics of queueing processes, like the busy period, are mainly represented by the lower levels.

As next step, we use the fact that high clipping levels lead to increased correlation for large time lags by introducing the approximation method presented in Section 9.4. To evaluate its impact, we choose the 33-level MAP as reference. It is of moderate size (134 states; which can be easily handled computationally) and is a good approximation to the exact solution. We now compare the 33-level MAP with a MAP *of the same size* but constructed as follows: (i) levels 0 and 1 are modeled without approximation, (ii) levels 2 through 63 are merged two by two, and (iii) the

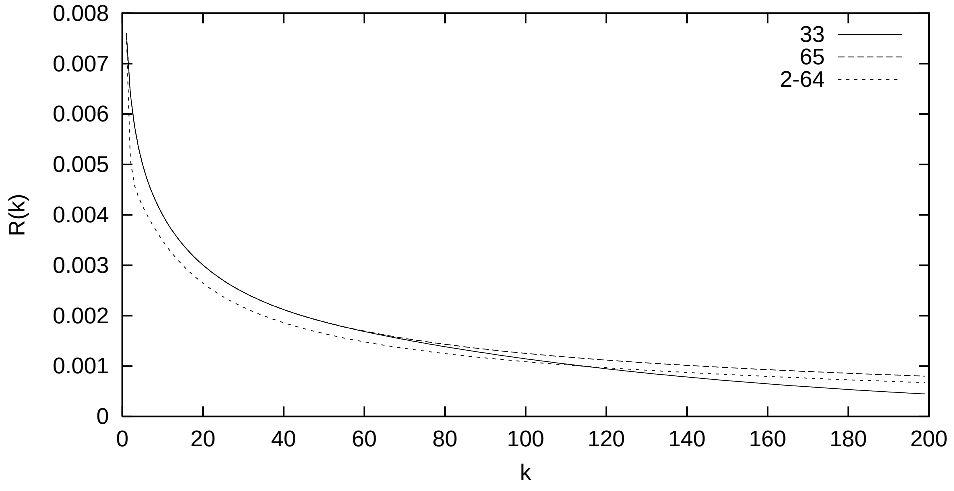


Figure 10.4: Autocovariance function $R(k)$ of the 2-64-level MAP

levels beyond 63 are merged into one level. This MAP (called 2-64-level MAP) is compared with the 33-level and the 65-level MAP in Figure 10.4. For $k \geq 110$ this MAP better approximates the exact solution than the 33-level MAP. However, for $k \leq 110$ its autocovariance is apparently lower than the 33-level solution: since the 2-64-level MAP approximates the levels from 2 through 63 two by two it gives a worse description of correlation between subsequent queue departures.

Even slightly better approximations can be obtained by using more complex level schemes, e.g., a 4-40-84-level MAP with the following structure: (i) levels 0 through 3 are modeled exactly, (ii) levels 4 through 39 are merged two by two and (iii) levels 40 through 83 are merged four by four (clipping level is 84). Figure 10.5 shows this MAP in comparison with the other approximations. Since the levels 0 through 3 are modeled exactly, this MAP better approximates the autocovariance for small k than the 2-64-level MAP. The autocovariance for large k is nearly identical for both MAPs.

10.2.2 Analysis results of MAP|MAP|1 tandem queues

In the previous section we have presented and compared different approximations to the output process of a PH|PH|1 queue. These approximations have the same inter-arrival time distribution but differ in higher order statistics, such as the autocovariance. It is well known that the presence of autocorrelation in arrival processes heavily impacts the performance measures of queueing systems [73]. Hence, in this section, we will examine how the different approximations affect the performance of a subsequent second queue which takes the (approximated) departure process of the first queue as arrival process.

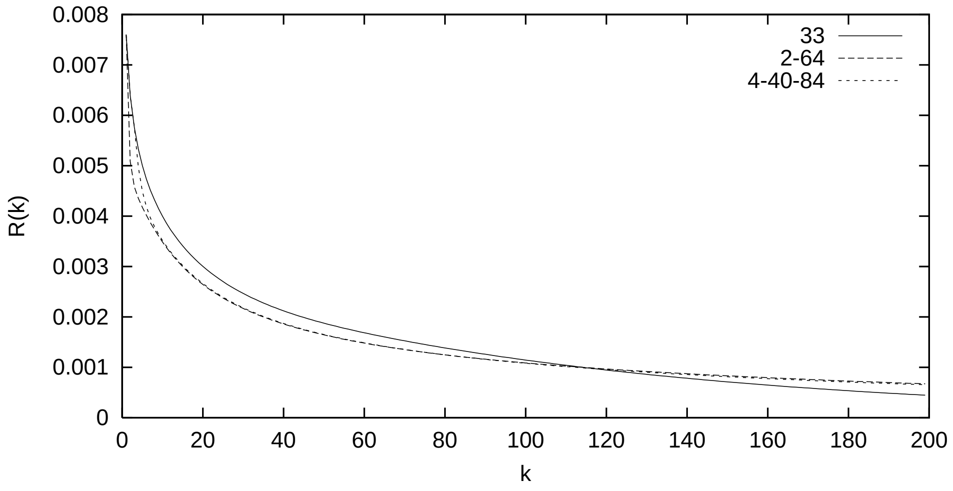


Figure 10.5: Autocovariance function $R(k)$ of the 4-40-84-level MAP

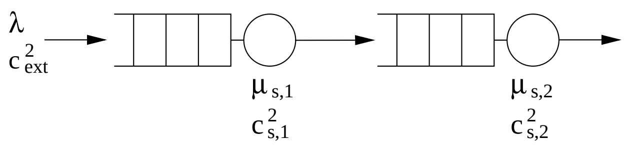


Figure 10.6: Queueing network for the approximation of the MAP|MAP|1 departure process

Hyper-exponential renewal input process

We keep the queueing station from the previous section and add a second queue with the same service process, i.e., $\lambda = 0.95$, $c^2_{ext} = 8.0$, $\mu_{s,1} = \mu_{s,2} = 1.0$, and $c^2_{s,1} = c^2_{s,2} = 0.5$ (see Figure 10.6). Table 10.3 shows the mean value of the second queue length as a function of the chosen approximated departure process, as well as the results of the simulation and of FiFiQueues. Additionally, we have given the results obtained using a PH renewal process with the same inter-departure time distribution as the departure process. The errors relative to the simulation are given in the middle column. The right column gives the number of states of the employed MAP, respectively, the (fitted) PH process.

The results show that the large SCV of the arrival time distribution, combined with the hypo-exponential service time distribution of the first station, yields a strongly correlated traffic stream arriving at the second station. As reported in other publications, the mean queue length of the second queue is significantly underestimated if the correlations are ignored, as it is the case for FiFiQueues and the PH process. A better approximation of the autocovariance function results in a higher mean queue length. This can be explained in a quite intuitive manner by looking at the power spectrum $P(w)$ (see also Appendix A) of the discrete function $f(k) = T_k$ where T_k is the k-th inter-arrival time of the first queue's departure

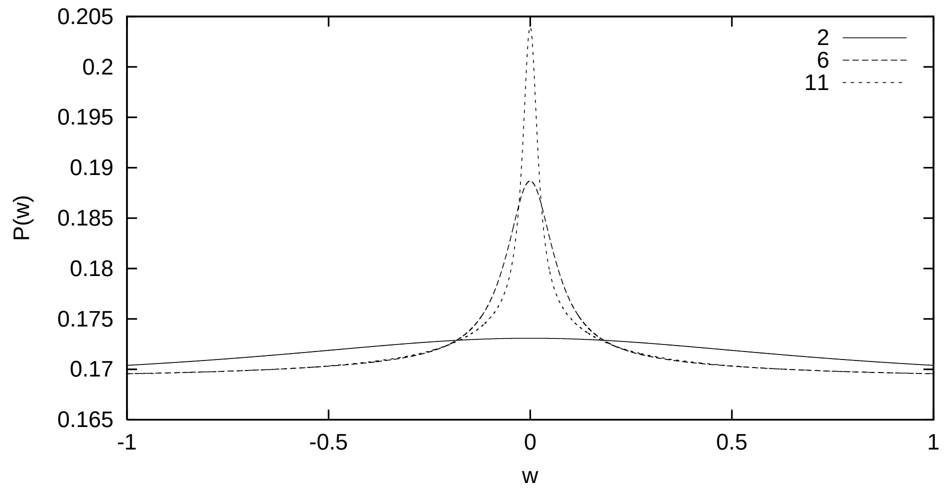


Figure 10.7: The power spectrum of the departure MAP for three different clipping levels

MAP. In Figure 10.7, we show the power spectrum for clipping levels 2, 6 and 11; we see that low-frequency energy, i.e., around $w = 0$, increases with the clipping level. This is an important fact, since San-qi Li has shown in [71] that input power in the low-frequency band has a dominant impact on queueing performance. Low frequencies produce long periods of increased workload entering the queue [73].

This example also shows that better approximations do not necessarily require more states, as becomes clear from the last row: a 2-40-120-level MAP with *168* states yields a better estimation of the mean queue length than a 51-level MAP with *208* states. This is a significant improvement since the complexity of MAP|MAP|1 queue analysis cubically depends on the number of states in the MAP describing the arrival process. It seems that the good modeling of correlation for large time lags, as provided by the 2-40-120-level MAP, is more important than for small time lags. This observation confirms the results of studies made in the area of self-similar traffic which report that correlation over many time scales may noticeably affect the performance of queueing systems [42].

However, a better approximation of the large time lags only improves the results when the first queue is run with a high load. Table 10.4 shows the results for a reduced external arrival rate of 0.8, i.e., when the load has been lowered from 0.95 to 0.8. For Table 10.5, we have additionally lowered the service rate of the second queue from 1.0 to 0.9. We observe in both cases that the best results are obtained by the approximations that focus on the small time lags.

We have also repeated the experiments with a hyper-exponential service time distribution at the second queue. Table 10.6, 10.7, and 10.8 show the results for the mean queue length when $c_{s,2}^2$ has been set to 2.0. Again, we observe that the complex approximations considerably improve the results for high loads (for the same number of states in the generated MAPs).

	E[N]	rel. error	#states
Simulation	23.9		
FiFiQueues	16.8	-29.7%	2
PH	14.4	-39.7%	8
31	20.5	-14.2%	128
41	21.5	-10.0%	168
51	22.4	-6.3%	208
61	22.9	-4.2%	248
2-60	21.5	-10.0%	128
2-80	22.3	-6.7%	168
2-40-80	21.7	-9.2%	128
2-40-120	22.5	-5.9%	168

Table 10.3: Mean queue length at the second queue for different approximations (tandem queues with hyper-exponential renewal input process, arrival rate 0.95, $\mu_2 = 1.0$, $c_{s,2}^2 = 0.5$)

	E[N]	rel. error
Simulation	5.70	
FiFiQueues	6.92	21.4%
21	5.59	-1.9%
31	5.69	-0.2%
10-30	5.57	-2.3%
2-60	5.49	-3.7%

Table 10.4: Mean queue length at the second queue for different approximations (tandem queues with hyper-exponential renewal input process, arrival rate 0.8, $\mu_2 = 1.0$, $c_{s,2}^2 = 0.5$)

	E[N]	rel. error
Simulation	20.7	
FiFiQueues	14.8	-28.5%
31	19.0	-8.2%
41	19.6	-5.3%
20-42	19.1	-7.7%
2-60	17.9	-13.5%

Table 10.5: Mean queue length at the second queue for different approximations (tandem queues with hyper-exponential renewal input process, arrival rate 0.8, $\mu_2 = 0.9$, $c_{s,2}^2 = 0.5$)

	E[N]	rel. error
Simulation	45.7	
FiFiQueues	30.4	-33.5%
31	37.9	-17.1%
41	39.5	-13.6%
51	40.8	-10.7%
61	41.8	-8.5%
2-60	39.6	-13.3%
2-80	41.0	-10.3%
2-40-80	40.0	-12.5%
2-40-120	41.2	-9.8%

Table 10.6: Mean queue length at the second queue for different approximations (tandem queues with hyper-exponential renewal input process, arrival rate 0.95, $\mu_2 = 1.0$, $c_{s,2}^2 = 2.0$)

	E[N]	rel. error
Simulation	9.93	
FiFiQueues	9.55	-3.8%
21	9.44	-4.9%
31	9.64	-2.9%
41	9.73	-2.0%
20-42	9.64	-2.9%
2-60	9.32	-6.1%

Table 10.7: Mean queue length at the second queue for different approximations (tandem queues with hyper-exponential renewal input process, arrival rate 0.8, $\mu_2 = 1.0$, $c_{s,2}^2 = 2.0$)

	E[N]	rel. error
Simulation	28.3	
FiFiQueues	20.4	-27.9%
31	26.2	-7.4%
41	26.9	-4.9%
51	27.2	-3.9%
2-60	24.9	-12.0%
2-80	25.1	-11.3%
2-40-80	24.9	-12.0%

Table 10.8: Mean queue length at the second queue for different approximations (tandem queues with hyper-exponential renewal input process, arrival rate 0.8, $\mu_2 = 0.9$, $c_{s,2}^2 = 2.0$)

	E[N]	rel. error
Simulation	27.7	
FiFiQueues	24.1	-13.0%
33	26.4	-4.7%
4-40-84	27.4	-1.1%

Table 10.9: Mean queue length at the second queue for different approximations (tandem queues with Poisson arrivals)

	E[N]	rel. error
Simulation	20.2	
33	18.9	-6.4%
2-64	19.2	-4.9%
4-40-84	19.4	-4.0%

Table 10.10: Mean queue length at the second queue for different approximations (tandem queues with non-renewal input process)

Experiments with other input processes

Comparably good results (as before) are also obtained with true Markovian arrival processes. Again, we assume two queues with Erlang-2 service and service rate 1. Table 10.9 shows the mean queue length at the second queue, obtained by feeding a Poisson process with arrival rate 0.98 into the first queue. The mean queue length of the first queue is 35.1. Again, our results are rather good.

The next arrival process is the superposition of a hyper-exponential renewal process with arrival rate 0.74 and SCV 8.0, with an Erlang-2 process with arrival rate 0.21. The thus-obtained process has an arrival rate of 0.95 and a SCV of 2.2. Since this process is not a renewal process, its limiting index of dispersion for counts considerably differs from its squared coefficient of variation: $I = 6.3$. The mean queue length of the first queue is 58.2. The results for the mean queue length of the second queue are shown in Table 10.10. Although slightly worse than before, the results are still acceptable. As in the previous experiments, the better results for the more complex approximations indicate that it is important to correctly model the correlations for large time lags.

10.2.3 Queueing networks with traffic splitting

Using MAPs for the traffic splitting operation in a queueing network is especially attractive, since, unlike the traffic merging operation, the resulting MAPs have the same number of states as the input MAP (see Section 3.1.3). FiFiQueues' splitting operation assumes that the involved processes are renewal processes (Section 5.3.4), hence we can expect an improvement. To test the traffic splitting we take again a

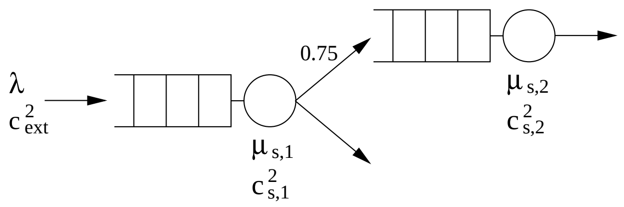


Figure 10.8: Queueing network with traffic splitting

tandem network with two queues, but this time only 75% of the departure traffic of the first queue is fed to the second queue, i.e., the splitting probability is 0.75 (see Figure 10.8). The external arrival traffic to the first queue has a rate of $\lambda = 1.2$. The SCV of the arrival time distribution is given by c_{ext}^2. The service rate of the first queue and the second queue is 1.5, resp. 1. The SCV of the service time distribution is $c_{s,1}^2$ for the first queue, respectively $c_{s,2}^2$ for the second queue. Table 10.11 and 10.12 show the results for the mean queue length for various values of c_{ext}^2, $c_{s,1}^2$, and $c_{s,2}^2$. The results are similar to those for the network without splitting. Note that we have chosen a not so extreme load for the first queue ($\rho_1 = 0.8$) to test the splitting operation under more realistic conditions. As consequence, we have omitted the results for the complex approximation schemes since they do not provide any further improvement.

10.2.4 Summary

In this section we have investigated the departure process of a MAP|MAP|1 and its approximation by a finite MAP. We have shown that the simple method presented in Section 9.3 provides a good approximation to the autocovariance function of the departure process. The more complex method presented in Section 9.4 is able to better approximate the autocovariance function for large time lags by MAPs with the same number of the states. The experiments have shown that the approximations allow us to accurately evaluate the performance of a subsequent queueing station. It depends on the sensitivity of the station to correlations with large time lags whether the complex approximations provide better results than the simple approximations.

10.3 Approximations for MAP|MAP|1|K queues

In this section, we evaluate the level condensing method introduced in Section 9.4 for queues with finite capacity. As test model, we reuse the tandem queueing network with hyper-exponential arrival process (SCV 8) from Section 10.2.2 but limit the capacity of the first queue to 10 (see Figure 10.9). This modification affects the autocovariance of the departure process of the first queue as shown in Figure 10.10 for an external arrival rate λ of 0.9 and an Erlang-2 service process; the figure

$c_{s,1}^2$	c_{ext}^2	method	E[N]	rel. error
0.5	0.5	Simulation	11.3	
		FiFiQueues	11.5	1.8%
		10	11.4	0.9%
		20	11.4	0.9%
	4.0	Simulation	19.8	
		FiFiQueues	15.7	-20.7%
		10	18.4	-7.1%
		20	19.4	-2.0%
		30	19.7	-0.5%
2.0	0.5	Simulation	12.8	
		FiFiQueues	14.6	14.1%
		10	12.9	0.8%
		20	12.7	-0.8%
	4.0	Simulation	21.1	
		FiFiQueues	18.3	-13.3%
		10	20.2	-4.3%
		20	20.6	-2.4%
		30	20.8	-1.4%
		50	20.9	-0.9%

Table 10.11: Mean queue lengths at the second queue with traffic splitting and $c_{s,2}^2 = 2$

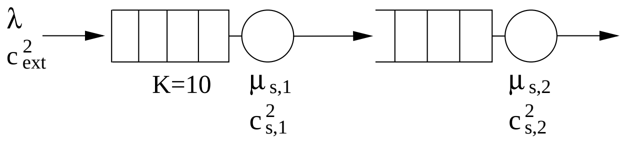


Figure 10.9: Queueing network for the approximation of the MAP|MAP|1|K departure process

$c^2_{s,1}$	c^2_{ext}	method	E[N]	rel. error
0.5	0.5	Simulation	4.14	
		FiFiQueues	4.26	2.9%
		10	4.15	0.2%
		20	4.15	0.2%
	4.0	Simulation	11.4	
		FiFiQueues	8.3	-27.2%
		10	10.1	-11.4%
		15	10.6	-7.0%
		20	10.9	-4.4%
2.0	0.5	Simulation	6.0	
		FiFiQueues	7.3	21.7%
		10	6.1	1.7%
		20	6.1	1.7%
	4.0	Simulation	13.0	
		FiFiQueues	10.8	-16.9%
		10	12.4	-4.6%
		20	12.7	-2.3%

Table 10.12: Mean queue lengths at the second queue with traffic splitting and $c^2_{s,2} = 0.2$

displays the autocovariance function of the exact departure process and of various approximations where "2–9" denotes the MAP obtained by condensing levels 2–9 of the original departure process, "3–8" stands for the MAP with condensed levels 3–8, et cetera. Note the peak in the autocovariance where the time lag k equals the queueing capacity $K = 10$. As can be seen, the shape of the autocovariance function is more distorted when we increase the number of condensed levels. The different MAPs have the following number of states:

MAP	states
exact	44
2-9	16
3-8	24
4-7	32
5-6	40

To test the quality of the approximation we compute the mean queue length E[N] of the second queue for different external arrival rates λ and different SCVs $c^2_{s,1}$ and $c^2_{s,2}$ of the two service processes. The results are shown in Table 10.13 and 10.14. The tables also contain the results obtained by FiFiQueues with two-moment descriptors, by three-moment descriptors (with Minimal Acyclic PH distributions; see Section 5.5.5), and by a PH renewal process that has the same inter-departure time

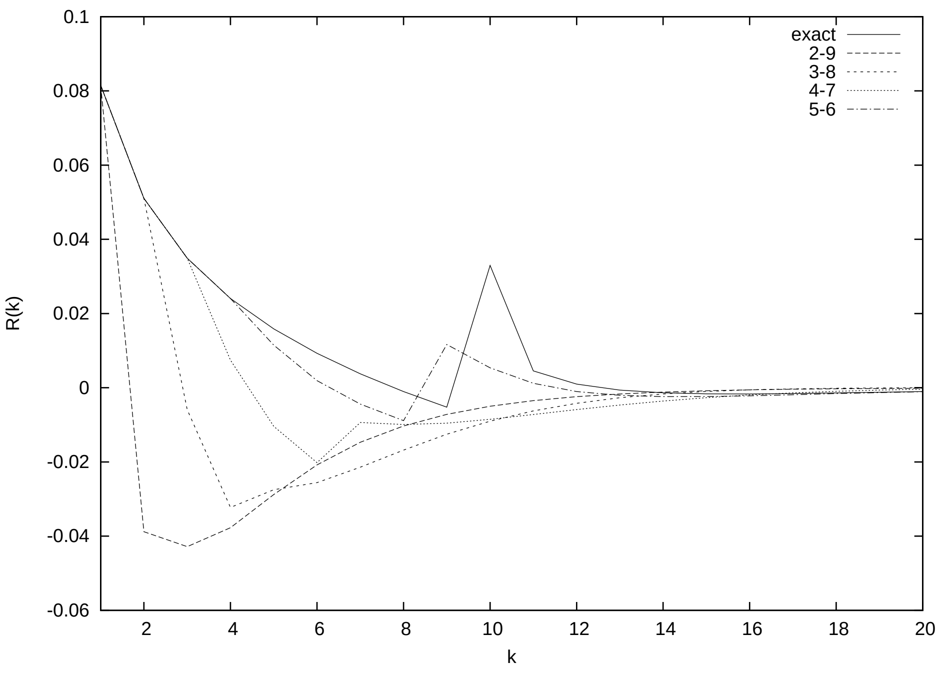


Figure 10.10: Autocovariance function $R(k)$ for approximations of the finite capacity queue ($\lambda = 0.9, c_{s,1}^2 = 0.5$)

distribution as the departure MAP.

In contrast to the experiments in Section 10.2, we observe this time that the PH renewal process provides good results with relative errors less than 7%. Hence, the errors obtained using two-moment and three-moment descriptors are clearly caused by the employed PH-fitting methods. The gain of the level condensing method depends on how important the autocovariance is for the performance of the second queue. The 3–8 MAP seems to be a good trade-off between quality of the results and size (number of states) in this case.

10.4 Approximation of processes with positive, exponentially decreasing autocovariance

We reconsider the tandem queueing network with hyper-exponential renewal input introduced in Section 10.2.2 and fit a MAP to the departure process of the first queueing station by using the method from Section 9.5. The fitted MAP is then fed as arrival process to the second queue.

We begin with the case with arrival rate= 0.95, $\mu_2 = 1.0$, $c_{s,2}^2 = 0.5$. It shows that the quality of the approximations depends on the value of p in the fitting procedure since the autocovariance function $R(k)$ of the departure process decays

λ	method	E[N]	rel. error
0.5	Exact	1.24	
	FiFiQueues	1.69	36.3%
	3 moments	1.44	16.1%
	PH	1.16	-6.5%
	2-9	1.18	-4.8%
	3-8	1.21	-2.4%
	4-7	1.23	-0.8%
	5-6	1.23	-0.8%
	($\rho_2 = 0.50$)		
0.7	Exact	2.26	
	FiFiQueues	3.46	53.1%
	3 moments	2.60	15.0%
	PH	2.13	-5.8%
	2-9	2.14	-5.3%
	3-8	2.19	-3.1%
	4-7	2.22	-1.8%
	5-6	2.25	-0.4%
	($\rho_2 = 0.64$)		
0.9	Exact	2.97	
	FiFiQueues	4.58	54.2%
	3 moments	3.44	15.8%
	PH	2.88	-3.0%
	2-9	2.86	-3.7%
	3-8	2.90	-2.4%
	4-7	2.93	-1.3%
	5-6	2.96	-0.3%
	($\rho_2 = 0.73$)		

λ	method	E[N]	rel. error
0.5	Exact	1.31	
	FiFiQueues	1.66	26.7%
	3 moments	1.43	9.2%
	PH	1.26	-3.8%
	2-9	1.28	-2.3%
	3-8	1.29	-1.5%
	4-7	1.30	-0.8%
	5-6	1.31	0%
	($\rho_2 = 0.49$)		
0.7	Exact	2.41	
	FiFiQueues	3.38	40.2%
	3 moments	2.60	7.9%
	PH	2.32	-3.7%
	2-9	2.33	-3.3%
	3-8	2.36	-2.1%
	4-7	2.39	-0.8%
	5-6	2.40	-0.4%
	($\rho_2 = 0.64$)		
0.9	Exact	3.24	
	FiFiQueues	4.56	40.7%
	3 moments	3.50	8.0%
	PH	3.18	-1.9%
	2-9	3.17	-2.2%
	3-8	3.19	-1.5%
	4-7	3.21	-0.9%
	5-6	3.23	-0.3%
	($\rho_2 = 0.73$)		

Table 10.13: Mean queue lengths for different approximations (tandem queues, first queue with finite capacity). Left table: $c_{s,1}^2 = 0.25, c_{s,2}^2 = 0.5$. Right table: $c_{s,1}^2 = 0.5, c_{s,2}^2 = 0.5$. ρ_2 gives the utilization of the second station.

λ	method	E[N]	rel. error
0.5	Exact	1.53	
	FiFiQueues	1.53	0%
	3 moments	1.41	-7.8%
	PH	1.54	0.7%
	2-9	1.55	1.3%
	3-8	1.55	1.3%
	4-7	1.55	1.3%
	5-6	1.54	0.7%
	($\rho_2 = 0.48$)		
0.7	Exact	2.86	
	FiFiQueues	3.08	7.7%
	3 moments	2.71	-5.2%
	PH	2.90	1.4%
	2-9	2.92	2.1%
	3-8	2.92	2.1%
	4-7	2.90	1.4%
	5-6	2.87	0.3%
	($\rho_2 = 0.61$)		
0.9	Exact	4.15	
	FiFiQueues	4.54	9.4%
	3 moments	4.05	-2.4%
	PH	4.24	2.2%
	2-9	4.24	2.2%
	3-8	4.22	1.7%
	4-7	4.19	1.0%
	5-6	4.16	0.2%
	($\rho_2 = 0.71$)		

λ	method	E[N]	rel. error
0.5	Exact	2.10	
	FiFiQueues	2.13	1.4%
	3 moments	2.01	-4.3%
	PH	2.10	0%
	2-9	2.12	1.0%
	3-8	2.13	1.4%
	4-7	2.12	1.0%
	5-6	2.11	0.5%
	($\rho_2 = 0.48$)		
0.7	Exact	3.92	
	FiFiQueues	4.12	5.1%
	3 moments	3.81	-2.8%
	PH	3.93	0.3%
	2-9	3.96	1.0%
	3-8	3.96	1.0%
	4-7	3.95	0.5%
	5-6	3.93	0.3%
	($\rho_2 = 0.61$)		
0.9	Exact	5.74	
	FiFiQueues	6.07	5.7%
	3 moments	5.69	-0.9%
	PH	5.81	1.2%
	2-9	5.78	0.7%
	3-8	5.79	0.9%
	4-7	5.76	0.3%
	5-6	5.74	0%
	($\rho_2 = 0.71$)		

Table 10.14: Mean queue lengths for different approximations (tandem queues, first queue with finite capacity). Left table: $c_{s,1}^2 = 2, c_{s,2}^2 = 0.5$. Right table: $c_{s,1}^2 = 2, c_{s,2}^2 = 2$. ρ_2 gives the utilization of the second station.

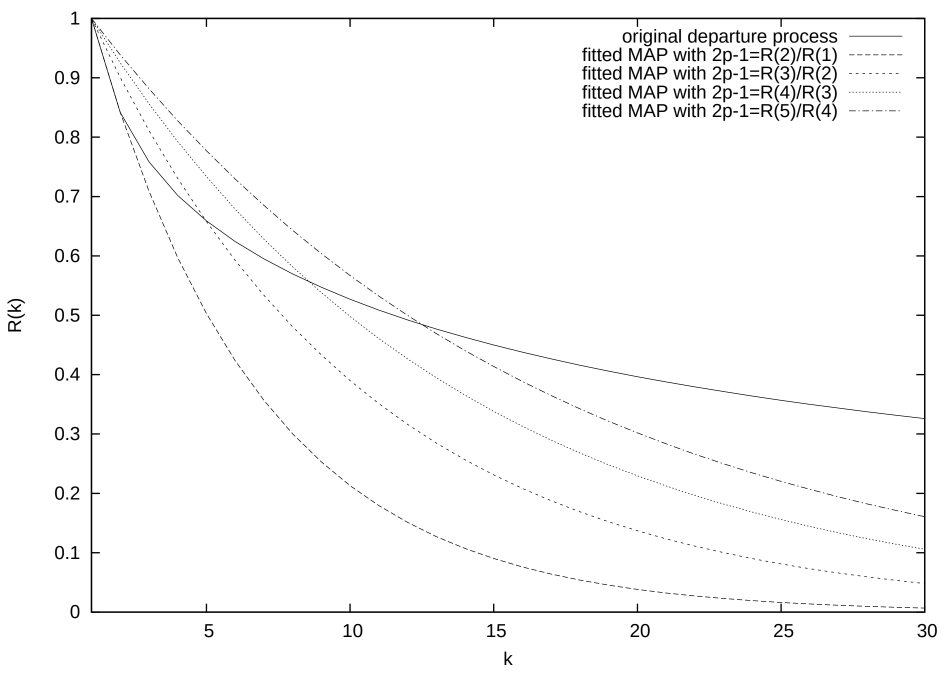


Figure 10.11: Normalized autocovariance function $R(k)$ as function of the time lag k for different approximations

slower than exponentially. Figure 10.11 shows the (normalized) $R(k)$ in comparison to the autocovariance function of the fitted MAP for various values of $2p-1$ with $r = \frac{R(1)}{2p-1}$. The resulting mean queue lengths of the second station are given in Table 10.15. We have also added results from other approximation methods. The row "PH" depicts the results obtained by a PH renewal process that has the same inter-departure time distribution as the departure MAP (see Section 9.6). Again, the right column shows the sizes of the involved MAPs, respectively PH process. The results for $c^2_{s,2} = 2$ can be found in Table 10.16.

As can be seen, the fitting approach is not so good as the methods based on level clipping if the value of $2p-1$ is derived from the autocovariance at small time lags. However, the fitting approach provides much better results than FiFiQueues and the PH renewal process although the fitted MAP only consists of 5 states. Especially the comparison to the PH process shows that the approximation of the autocovariance function is more important than an exact reproduction of the distribution function (at least for *this* network configuration; the experiments in Chapter 6 and Section 10.3 illustrate that the autocovariance does not always play a dominant role in the performance of a queueing station). We observe that, within the fitted MAPs, the best results are obtained with $2p-1 = R(6)/R(5)$ because the resulting MAP better approximates the autocovariance for large time lags.

Of course, the method fails if it is applied to point processes with an autocovariance function of different, non-exponential shape. Beside of the obvious case

	E[N]	rel. error	#states
Simulation	23.9		
FiFiQueues	16.8	-29.7%	2
PH	14.4	-39.7%	8
31	20.5	-14.2%	128
41	21.5	-10.0%	168
51	22.4	-6.3%	208
2-80	22.3	-6.7%	168
2-40-120	22.5	-5.9%	168
Fitted MAP with:			
$2p-1=R(2)/R(1)$	18.1	-24.3%	5
$2p-1=R(3)/R(2)$	19.7	-17.6%	5
$2p-1=R(4)/R(3)$	21.1	-11.7%	5
$2p-1=R(5)/R(4)$	22.3	-6.7%	5
$2p-1=R(6)/R(5)$	23.3	-2.5%	5

Table 10.15: Mean queue length of the second queue for different approximations (hyper-exponential input process, arrival rate 0.95, $\mu_2 = 1$, $c_{s,2}^2 = 0.5$)

	E[N]	rel. error	#states
Simulation	45.7		
FiFiQueues	30.4	-33.5%	2
PH	28.9	-36.8%	8
41	39.5	-13.6%	168
51	40.8	-10.7%	208
61	41.8	-8.5%	248
2-80	41.0	-10.3%	168
2-40-120	41.2	-9.8%	168
Fitted MAP with:			
$2p-1=R(2)/R(1)$	32.1	-29.8%	5
$2p-1=R(3)/R(2)$	33.7	-26.3%	5
$2p-1=R(4)/R(3)$	35.0	-23.4%	5
$2p-1=R(5)/R(4)$	36.2	-20.8%	5
$2p-1=R(6)/R(5)$	37.2	-18.6%	5

Table 10.16: Mean queue length of the second queue for different approximations (hyper-exponential input process, arrival rate 0.95, $\mu_2 = 1$, $c_{s,2}^2 = 2$)

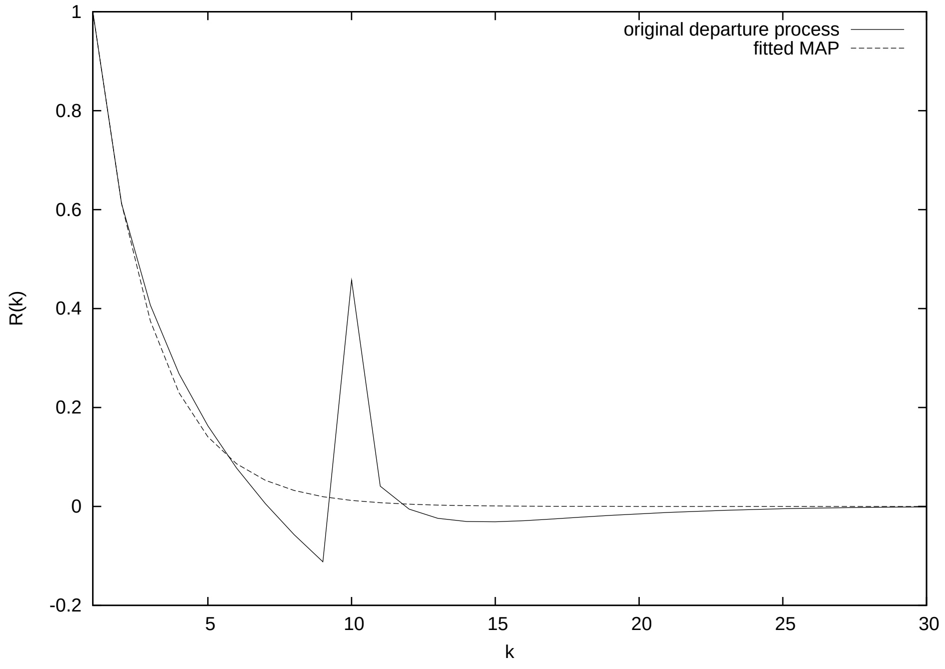


Figure 10.12: Normalized autocovariance function $R(k)$ of the departure process of the finite queueing station

where $R(k) < 0$ for all $k > 0$, more complex shapes can arise. Figure 10.12 shows $R(k)$ of the departure process if we specify a finite capacity of 10 for the first queue of our tandem queueing network with arrival rate= 0.95. Here, our method (with $2p - 1 = R(2)/R(1)$) is only able to fit the positive decreasing part of the autocovariance function and not its peak where the time lag k equals the queueing capacity of the first station. Table 10.17 shows the resulting mean queue lengths for $\mu_2 = 1.0$, $c^2_{s,2} = 0.5$. The row labeled "Exact" in the table has been computed by directly using the departure MAP of the first queue as arrival MAP of the second queue without any approximation. The rows "2–9", "3–8", "4–7" and "5–6" give the results obtained by the level condensing method, as discussed in Section 10.3. As expected, the fitted MAP performs better than FiFiQueues but worse than the other approximations.

10.5 Summary and conclusions

In this chapter, we have evaluated the performance of our approximation and reduction techniques for MAP-based traffic descriptors. The experiments have shown that the methods provide very good results when the considered MAPs more or less fulfill certain, method-specific, conditions. The reduction by removing equivalent

	E[N]	rel. error
Exact	3.42	
FiFiQueues	4.78	39.8%
PH	3.37	-1.5%
2-9	3.35	-2.0%
3-8	3.37	-1.5%
4-7	3.38	-1.2%
5-6	3.41	-0.3%
Fitted MAP	4.49	31.3%

Table 10.17: Mean queue length for the finite capacity model (hyper-exponential input process, arrival rate 0.95, $\mu_2 = 1.0$, $c_{s,2}^2 = 0.5$)

states can only be applied to the very restricted class of MAPs generated by the superposition of two nearly identical departure processes. This is caused by the regular structure of the departure MAPs where each state has its unique meaning. The finite output process approximation for MAP|MAP|1 queues and the level condensing method for the output of MAP|MAP|1(|K) are of special interest because they preserve the first-order characteristics (inter-departure time distribution) of the departure process. They can provide a compact approximation of the original process with a good reproduction of the autocovariance. The last method, the fitting to a MAP with exponentially decreasing autocovariance, is not restricted to queue departure processes and yields very small MAPs with a good (three-moment based) approximation to the inter-event time distribution. Naturally, the results are only satisfying if the requirements to the shape of the autocovariance function are met.

Although the presented reduction methods provide very good results, it should be noted that their application is currently limited to small, cycle-free networks since the achieved reduction only decreases the effects of the state space explosion for a few station analysis steps, but does not totally avoid them. For an iteration-based decomposition analysis of a large network the reduction methods must be combined with other methods such as [48].

Chapter 11

Conclusion

In this thesis, we have presented a decomposition framework for the analysis of a fairly general class of open and closed queueing networks. The decomposition is done at queueing station level, i.e., the queueing stations are independently analyzed. During the analysis, traffic descriptors are exchanged between the stations, representing the streams of jobs flowing between them. The analysis of a queueing station is divided into fundamental steps that describe how the incoming traffic streams are merged, processed at the queueing station, and then split into the outgoing traffic streams. Networks with feedback are analyzed using a fixed-point iteration.

Based on the decomposition framework, we have developed an analysis method called *FiFiQueues*. The method supports open queueing networks with infinite and finite capacity queues. We have been able to prove the existence of the fixed point for the employed fixed-point iteration. Our experiments have shown that FiFiQueues is easy usable, efficient and provides good results for important performance measures, like mean queue length, even if the network has a complex structure. However, the experiments have also shown that larger errors have to be expected if the traffic streams inside the analyzed network exhibit strong correlations.

FiFiQueues allows arbitrary phase-type renewal processes as service processes. We have describe how hyper-exponential distributions, which are a special case of phase-type distributions, can be fitted to heavy-tail distributed measurement data using the EM-algorithm. Additionally, we have presented an extension that applies a stratification approach to the data in order to increase the efficiency of the method.

Like in QNA [115], the traffic descriptors used in FiFiQueues are based on the first and second moment. We have discussed the possibility to extend the traffic descriptors of FiFiQueues to three moments. Since FiFiQueues transforms the traffic descriptors to phase-type renewal processes during the analysis, a fitting method for three moments to phase-type distributions is required. We have conducted experiments with three existing fitting procedures, however, it appears that three-moment descriptors do not significantly improve the results in queueing networks with feedback. Since they considerably increase the runtime of the analysis, we currently refrain from using three-moment descriptors in FiFiQueues until further examinations.

We have also proposed a decomposition-based method for the analysis of closed queueing networks. It is especially attractive because it is founded on existing analysis methods for open networks. Our implementation with FiFiQueues suggests that the method is able to provide useful results for a broad class of closed networks. Additionally, the method is very fast even for large networks and populations, but we have also shown that it only works well if the analyzed network contains exactly one bottleneck.

The FiFiQueues algorithm for open and closed queueing networks as well as a fast discrete-event simulator have been implemented in an integrated tool with a graphical user interface that allows to construct, edit and evaluate queueing networks of arbitrary topology.

FiFiQueues' traffic descriptors cannot account for correlations in the traffic streams. To approach this problem, we have introduced MAPs as traffic descriptors. However, a queueing network analysis based on MAP traffic descriptors and MAP|MAP|1(|K) queues suffers under the state space explosion problem, since the involved operations increase the size of the descriptors in each iteration. Hence, we have presented five different MAP reduction methods in order to remove or decrease the effect of the state space explosion. The experiments have shown that the methods provide very good results when the considered MAPs more or less fulfill certain, method-specific, conditions. However, we have also observed that the achieved reduction only decreases the effects of the state space explosion, but does not totally avoid them. Hence, the application of our reduction methods is currently limited to small, cycle-free networks.

As for future work, we envisage to embed the presented reduction algorithms into a complete analysis algorithm for general queueing networks. If combined with methods such as [48], our algorithms could be used to accurately analyze those parts of the queueing network model that are known to be very sensitive to correlations in the traffic stream. Concerning FiFiQueues, we still believe that the extension of the FiFiQueues algorithm to three-moment descriptors is worthwhile. Future research has to show in which situations three-moment descriptors show a clear advantage over the two-moment descriptors used by FiFiQueues.

Finally, we want to mention that the FiFiQueues tool has been successfully used at a German system vendor of logistics software in the modeling and planning of the production lines of an order picking warehouse. Due to the confidential nature of that design, we cannot provide any further details on this in this thesis.

Appendix A

The Power Spectrum of a MAP

The power spectrum of the discrete process $T_1, T_2, \ldots$ is the Fourier transform of the autocovariance function $R(k) = \mathrm{E}\left[(T_1 - \mathrm{E}[T_1])(T_{k+1} - \mathrm{E}[T_{k+1}])\right]$. In the special case of a MAP $(\mathbf{Q_0}, \mathbf{Q_1})$, we have (see Section 3.1.2):

$$\begin{aligned} R(k) &= \mathrm{E}\left[(T_1 - \mathrm{E}[T_1])(T_{k+1} - \mathrm{E}[T_{k+1}])\right] \\ &= \mathbf{p}(-\mathbf{Q_0})^{-2}\mathbf{Q_1}\left\{\left[(-\mathbf{Q_0})^{-1}\mathbf{Q_1}\right]^{k-1} - \mathbf{1p}\right\}(-\mathbf{Q_0})^{-1}\mathbf{1}. \end{aligned}$$

The Fourier transform $\Phi(\omega)$ is given by [102]:

$$\Phi(\omega) = \sum_{k=-\infty}^{\infty} R(k)e^{-i\omega k}.$$

Since the process is weakly stationary, we have $R(-k) = R(k)$, which leads to

$$\begin{aligned} \Phi(\omega) &= R(0) + \sum_{k=1}^{\infty} R(k)(e^{i\omega k} + e^{-i\omega k}) \\ &= R(0) + 2\mathbf{C_1} \cdot \sum_{k=1}^{\infty} \left\{\left[(-\mathbf{Q_0})^{-1}\mathbf{Q_1}\right]^{k-1} - \mathbf{1p}\right\} \cos(\omega k) \cdot \mathbf{C_2}, \qquad \text{(A.1)} \end{aligned}$$

where $\mathbf{C_1} = \mathbf{p}(-\mathbf{Q_0})^{-2}\mathbf{Q_1}$ and $\mathbf{C_2} = (-\mathbf{Q_0})^{-1}\mathbf{1}$. The infinite sum in Equation (A.1) can be approximated by a finite sum with, e.g., $k = 1, \ldots, 100$. For optimal performance, it can be useful to switch to an eigenvalue based representation of the matrix $(-\mathbf{Q_0})^{-1}\mathbf{Q_1}$.

Appendix B

The Simulator

B.1 Introduction

In this appendix we describe the queueing network simulator that has been used to produce the simulation results presented in this thesis. Our goal is to provide an overview about the functionality of the simulator.

The simulator is a collection of C++-classes. From the beginning the simulator has been designed as platform for the efficient and rapid implementation of new experimental components like new probability distributions or new scheduling mechanisms. Hence, the C++-interface of the simulator has been optimized from the viewpoint of component developers and not for end users. We have tried to base the implementation as much as possible on a clear object-oriented design that supports code reusability. This rule has only been violated at some internal performance-critical code sections. The developed class library can be used in two ways:

1. To write simulation programs as ordinary C++-programs. That means that the data structures that represent the model are created by writing down by hand C++-source code based on the class library. The source code has to be compiled and linked to obtain an executable file which simulates the model and prints the results when executed. This approach has been followed for the example given in Section B.3.

2. To write an universal, user-friendly simulation program that accepts model specifications in form of, e.g., XML files or graphical input. This approach requires a front-end that reads the model specification and transforms it into data structures using the C++-classes of the simulator environment. Then the simulation is started and the obtained results are transformed back into a human-readable format. An example for this approach is the simulator included in the *FiFiQueues* network analyzer (see Chapter 8).

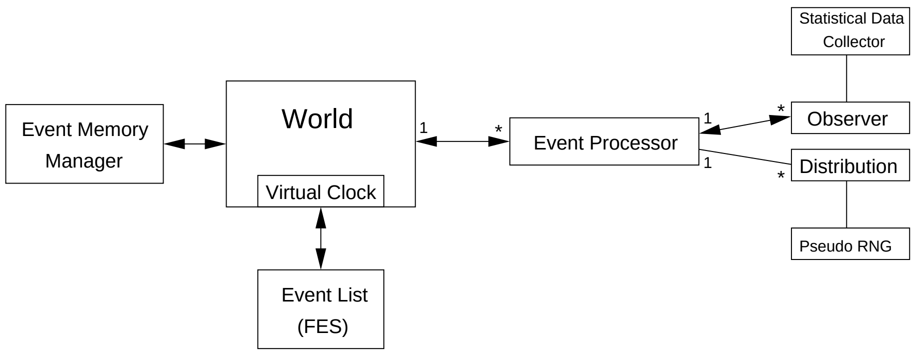


Figure B.1: Overview of the logical structure of the simulator

This appendix is further structured as follows: we first give an overview of the structure of the simulator in Section B.2 and briefly discuss the offered C++-classes. An example presented in Section B.3 shows how these classes are employed to simulate a small queueing network.

B.2 Overview

B.2.1 Logical structure

Figure B.1 shows an overview of the logical structure of the simulator. Lines with arrowheads represent event exchanges between components of the simulator whereas simple lines without arrowheads stand for general data exchange, e.g., state information.

The central component of the simulator is the *World* object. As any discrete-event simulator the simulation world has got a *virtual clock* showing the virtual time in the simulated scenario. The *event list* (or *future event set (FES)*) is a sorted list of the events that should be processed in the future. The *event processors* are the “workers” of the simulation world. Any event placed into the FES has a time stamp giving its execution time and a data field that specifies the destination event processor of the event. When a simulation runs, the world fetches the next event from the FES, as scheduled according to its time stamp, and sends it to its destination for execution. For optimal performance, event memory allocation and deallocation is managed by the *event memory manager*.

An event processor is the implementation of a model component, for example, a queueing station. Event processors are able to process (or execute) events and to send new events to other event processors (using the FES). The state of an event processor can be observed by one or more *observers*. An observer may be passive, i.e., only gathering data or it may be active. Active observers may influence the simulation run. Typical examples for passive observers are the *InterTimeObserver*

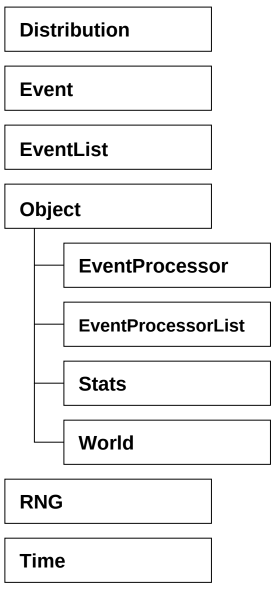


Figure B.2: Simulator core classes

(not shown in the figure) which compute statistics of inter-event times (for example inter-arrival times) using a *statistical data collector*. Active observers control the flow of simulation and react on changes in the state of the associated event processor. Usually a simulation model contains at least one such active observer that stops the simulation as soon as some user specified conditions are fulfilled. Event processors are supported by other objects (like *pseudo random number generators (pseudo RNGs)* or *distributions*) to generate new events according to user specifications.

B.2.2 Class organization of the implementation

Figures B.2 to B.7 show the organization of the classes of the C++-implementation. The trees with their roots and leaves stand for the inheritance relations between the classes — a leaf class inherits from the upper class in the tree.

The class structure deviates in some points from the logical structure described in the previous section. This is mainly for performance reasons: classes that are often instantiated are kept simple and without parent class to reduce overhead during instantiation.

Core classes

The core classes of the simulator are shown in Figure B.2.

Generally speaking, the job of the `Distribution` class (and its subclasses) is to feed the simulator components with sequences of numbers for the implementation of random point processes. Most of the subclasses of the `Distribution` class (shown in

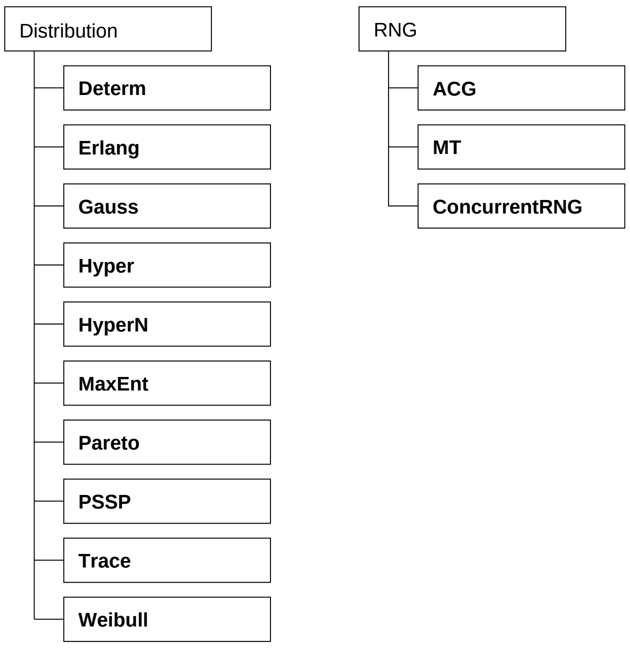


Figure B.3: Overview of distribution function and pseudo RNG classes

the left tree in Figure B.3) provide random numbers that are distributed according to specific distributions (hence the name of the super class) and some common distributions like deterministic, Erlang, Normal, etc., are included.

Most subclasses of the `Distribution` class need a pseudo random number generator (RNGs). The superclass of all RNG generators is the `RNG` class. The right tree in Figure B.3 shows three RNG implementations: the well known Acyclic Congruent Generator (`ACG`), the Mersenne Twister generator (`MT`), and an implementation of a thread-based concurrent RNG (`ConcurrentRNG`). The Mersenne Twister [80] is an excellent new random number generator which is much faster than the ACG and provides the astronomical period of $2^{19937} - 1$. We strongly encourage its usage since today's computer power is able to exhaust the period of 2^{32} or less of ordinary generators.

The `Event` class implements a generic event in the simulation world. Objects from this class contain a pointer to their destination (an instance of the `EventProcessor` class) and the delivery time (of type `Time`). Additionally, the `Event` class manages the memory allocation and deallocation of event objects.

"Live" objects in the simulation world are instances of subclasses of the `Object` class. This class defines an output method that is used by the simulator to generate a report when simulation ends. The only objects that are able to receive and process events are instances of the `EventProcessor` class and its subclasses.

The simulation world itself, class `World` is a subclass of `Object`, too. The world maintains the future event set (implemented as an `EventList`), the current virtual time, and a list of all `EventProcessor` objects in the world (class `EventProcessorList`). This list is used to broadcast important information about the state of the simulation, for example, when the warm-up phase ends.

The last core class, the `Stats` class is usually not used directly but by means of the observer objects described below.

Observers and statistics

Many event processors can experience changes in their internal state that the user may want to record statistically. For this purpose event processors maintain *observer lists* for every kind of state change and the user can add *observers* to these lists. If a state change occurs all observers in the associated list are notified. For example, a queueing station offers observer lists for job arrival times and queue length changes. The simulator currently provides two general classes of observers (see left tree in Figure B.4).

When a certain state change occurs, observers of the class `EventObserver` simply receive the event that triggered the state change as notification information. The simplest example is the `StopSim` class: it counts the received events and stops the simulation when a certain threshold has been reached. The `StatStarter` class is used to end the warm-up phase: it notifies all event processors about the end of the warm-up phase as soon as a certain number of events have been received. The thus-notified event processors may then decide to start the collection of statistical data, hence the name of the observer. The `InterTimeObserver` class can be used when one is interested in the time intervals between two received events.

The other class of observers, defined in `DataEventObserver`, receives a data item (e.g., the queue length) in addition to the trigger event. Instances of the `DataObserver` class are only interested in this data item, `DataInterTimeObserver` objects also respect the time intervals between the received data items.

Note that the observers do not compute any statistical information on their own. This is done by the `Stats` objects (shown in the right tree in Figure B.4). A user that adds an observer to a specific observer list only notifies the system that it is interested in the occurrence of the state change associated to the list. When the observer receives the trigger event (and the data item if it is a `DataEventObserver`) it has to pass the information to a `Stats` object. This object is chosen by the user depending on the amount of statistical information that should be computed. `Stats` objects of type `Moments` compute the first and second moment of the received data whereas `Moments3` also computes the third moments. The `ConcurrentMoments` class is a thread-based concurrent implementation of the `Moments` class.

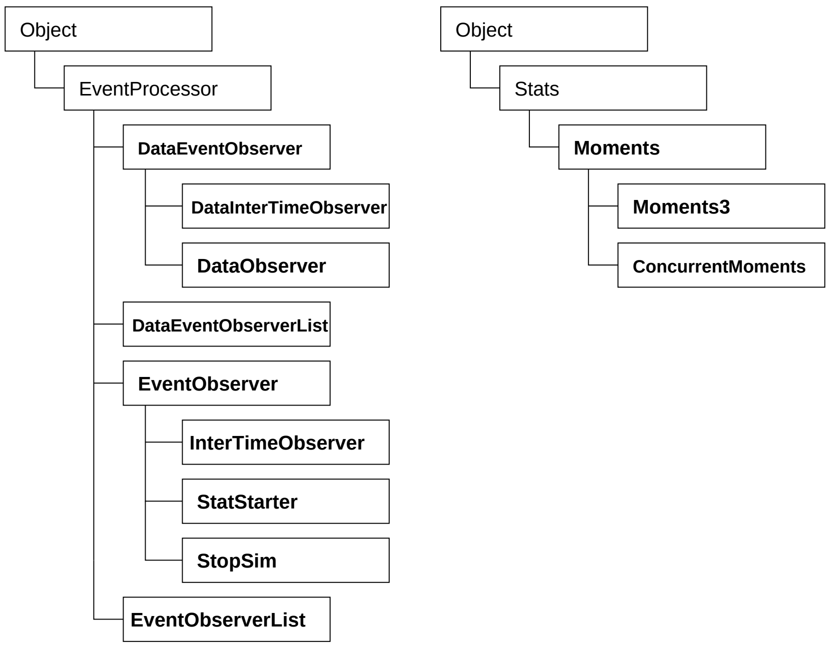


Figure B.4: Overview of observer and statistics classes

Queueing network components

The classes described above implement general concepts for the simulation of various scenarios. The following classes are more specific and have been developed for the simulation of communication networks, especially queueing networks. In our simulator a network consists of event generators and input processors (see Figure B.5). A generator (class `Generator`) continuously sends events to its destination, an input processor, where the length of the time interval between two events is controlled by a `Distribution` object. These events, we call them network events, represent the jobs traveling through the network. `InputProcessor` objects are special event processors that are able to receive these network events. Often, an input processor forwards the received network event to its destination after some processing delay.

The most basic input processor is the `Sink` object. It simply discards received events. The `ProbSplitter` resp. `TripleProbSplitter` forwards received events to one out of two resp. three different destinations which is probabilistically selected by an uniform distribution.

A more complex input processor is the `QueueingStation` class which is shown together with its helper classes in Figure B.6. This class implements a queueing station with a finite or infinite buffer and one or more service stations. Incoming network events are queued into the waiting buffer (class `Queue`) and then are scheduled for service in the service stations. The scheduling, i.e., the strategy to get the next event for service from queue, is controlled by a `Scheduling` object where the most popular is the FCFS scheduling. The service stations generate internal events

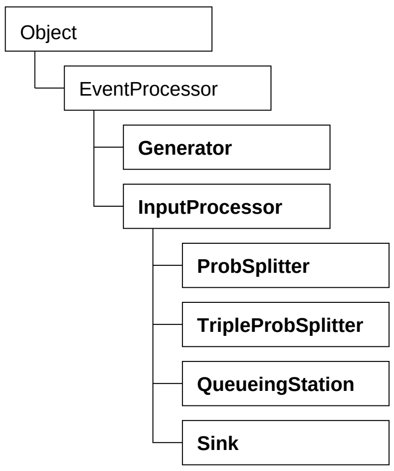


Figure B.5: Overview of network component classes

of type `ServiceEvent` to indicate the end of service time. The length of the service time for an network event is not simply given by a `Distribution` object. Instead, the network event is passed to a `ServiceTime` object which computes the service time for this event. We will see below why this intermediate step is done.

Events with class number

Until this point we have everything to simulate queueing networks. However, for complex network models it is often useful to have components that are able to process “classed” network events (prefix `Job` in the class hierarchy). When an event is generated a job class number is assigned to it. Then the input processors should give different services to the events according to this number.

Figure B.7 shows the classes required for the support of classed network events. Obviously, we need a new kind of event that extends the generic event by a job class number field. This is done by `JobEvent`. A new splitter is introduced in `JobClassSplitter` which does not randomly choose the destination but by the job class number of the received event. Instances of class `JobClassChanger` are used to change the job class number of an event. The service time in the queueing station is controlled by `JobServiceTime`. It allows the specification of different service time distributions (i.e, `Distribution` objects) for each job class.

B.3 Example

In this section we explain the simulator source code (see Figure B.8) for a small network consisting of two event generators that feed a queue with events of two different job classes.

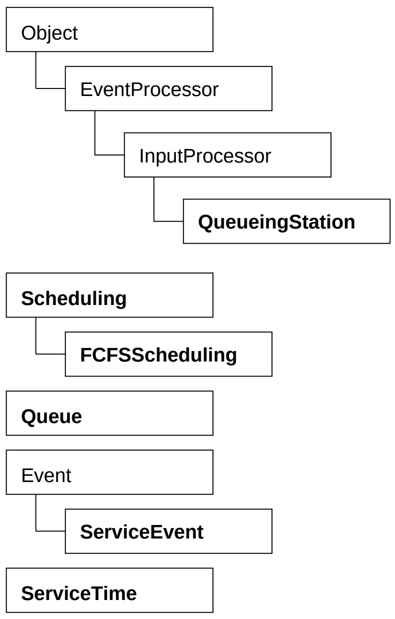


Figure B.6: Overview of queueing station classes

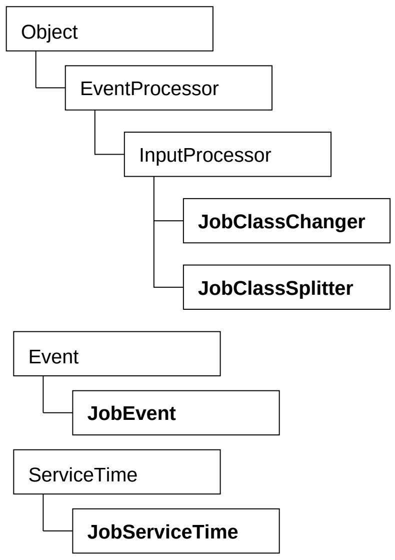


Figure B.7: Classes for support of events with class number

Before we can specify the network model, we have to declare and initialize two common data structures: the memory manager for the event class (line 9) and the world object (line 10). Additionally, all distributions used in the model should share one RNG (line 11). By default the RNG initializes itself ("seeding") using the current time.

The first event generator (lines 13–14) sends network events to the queue with job class number 0. The time between event generation should be hyper-exponentially distributed with mean 2.0 and variance 10.0. Since generators are very generic and do not know about special event subclasses we have to give to the generator an example of the events that we want to be generated.
The argument `InputProcessor_eventInput` specifies that the event should be send to the destination of the generator, the queue, as a network event, i.e., the queue should process the event by its input processor receipt-method, not by its general event processor receipt-method. In this way, the queue can differ between incoming network events and other events like system events. The second generator is defined in the same way, this time with job class number 1 (lines 15–16).

We continue with the queueing station. Its service station should process events of job class 0 with an Erlang-distributed service time with mean 1.0 and variance 0.5. Events of job class 1 should get Erlang-distributed service with mean 1.0 and variance 0.25. We define these two distributions in an array (line 17) which we will later pass to the `JobServiceTime` object of the queue. The queueing station (lines 18–19) should have one service process and an infinite queueing buffer (indicated by `-1`). Since the served jobs should leave the network we need a sink object (line 20).

Finally, we establish the connections between the network components (lines 22–23) and we are ready to start the simulation (line 34) and output the results (lines 36–39). However, note that the network components do not collect any statistical information by default, and the simulation will not terminate because no stop condition has been specified. Hence, we add some observers to the queueing station and the sink. We construct observers that collect information using `Moments` statistical objects (lines 25–30). Additionally, we assign an observer to the sink that stops the simulation as soon as 10^6 jobs have reached the sink (line 31). To allow the simulator to warm up, the statistical observers should not start their work before the sink has received $10,000$ network events. This is achieved by a `StatStarter` observer (line 32).

```
#include <iostream.h>
#include "World.h"
#include "rngs/MT.h"
#include "distributions/hyper.h"
#include "distributions/erlang.h"
...

int main(int argc, char *argv[]) {
  Event::init_mempool() ;
  World world("My World") ;
  MT rng ;

  Generator gen1("Generator 1",&world,new Hyper(2.0,10.0,&rng),
         new JobEvent(0,NULL,NULL,InputProcessor_eventInput,0)) ;
  Generator gen2("Generator 2",&world,new Hyper(2.0,10.0,&rng),
         new JobEvent(0,NULL,NULL,InputProcessor_eventInput,1)) ;
  Distribution *sd[]={new Erlang(1.0,0.5,&rng),new Erlang(1.0,0.25,&rng)} ;
  QueueingStation queue("My queueing station",&world,
         new FCFSScheduling(),new JobServiceTime(sd),1,-1) ;
  Sink sink("My sink",&world) ;

  gen.setDestination(&queue) ;
  queue.setDestination(&sink) ;

  queue.responseTimeObservers.push_back(
         new DataObserver(NULL,&world,new Moments())) ;
  queue.queueLengthObservers.push_back(
         new DataInterTimeObserver(NULL,&world,new Moments())) ;
  sink.inputEventObservers.push_back(
         new InterTimeObserver(NULL,&world,new Moments())) ;
  sink.inputEventObservers.push_back(new StopSim(NULL,&world,1000000)) ;
  sink.inputEventObservers.push_back(new StatStarter(NULL,&world,10000)) ;

  world.start() ;

  std::cout << endl ;
  std::cout << world ;
  std::cout << queue ;
  std::cout << sink ;

  return EXIT_SUCCESS ;
}
```

Figure B.8: Source code of the simulation example

Appendix C

Publications by the Author

C. Görg, R. Popp, M. Schmitt, D. Trossen, B.R. Haverkort, and R. Sadre. Data transmission in ATM networks: Applications, interfaces, protocols, and performance. In *6th IFIP Workshop on Performance Modelling and Evaluation of ATM Networks, Participants Proceedings*, 1998.

M. Hanus and R. Sadre. An abstract machine for Curry and its concurrent implementation in Java. *Journal of Functional and Logic Programming*, 1999(6), 1999.

[1]R. Sadre, B.R. Haverkort, and A. Ost. An efficient and accurate decomposition method for open finite- and infinite-buffer queueing networks. In W. Stewart and B. Plateau, editors, *Proceedings 3rd Int. Workshop on Numerical Solution of Markov Chains*, pages 1–20. Zaragosa University Press, 1999.

[1]R. Sadre and B.R. Haverkort. FiFiQueues: Fixed-point analysis of queueing networks with finite-buffer stations. In *MMB (Kurzvorträge)*, volume 99-16, pages 77–80. Universität Trier, 1999.

[1]R. Sadre and B.R. Haverkort. FiFiQueues: fixed-point analysis of queueing networks with finite-buffer stations. In *Computer Performance Evaluation. Modelling Techniques and Tools: 11th International Conference, TOOLS 2000*, volume 1786 of *Lecture Notes in Computer Science*, pages 324–327. Springer, 2000.

[2]R. Sadre and B.R. Haverkort. Characterizing traffic streams in networks of MAP/MAP/1 queues. In *Proceedings 11th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB 2001)*,

[1]These papers describe the original FiFiQueues algorithm (without extensions or proof of existence of the fixed point). Section 5.3 is based on these.

[2]This paper describes the finite output process approximation for MAP|MAP|1 queues as discussed in Section 9.3.

pages 195–208. VDE Verlag, 2001.

[3]R. El Abdouni Khayari, R. Sadre, and B.R. Haverkort. Fitting World-Wide Web request traces with the EM-algorithm. In *Proceedings of SPIE 4523 (Internet Performance and Control of Network Systems)*, pages 211–220, 2001.

R. El Abdouni Khayari, R. Sadre, and B.R. Haverkort. A validation of the pseudo self-similar traffic model. In *2002 International Conference on Dependable Systems and Networks (DSN 2002)*, pages 727–734. IEEE Computer Society, 2002.

R. El Abdouni Khayari, R. Sadre, and B.R. Haverkort. A class-based least-recently used caching algorithm for WWW proxies. In *Proceedings of the 2th Polish-German Teletraffic Symposium, Gdansk, Poland*, September 2002.

R. El Abdouni Khayari, R. Sadre, B.R. Haverkort, and N. Zoschke. Weighted fair queueing scheduling for World Wide Web proxy servers. In *Proceedings of SPIE 4865 (Internet Performance and Control of Network Systems III)*, pages 120–131, 2002.

[3]R. El Abdouni Khayari, R. Sadre, and B.R. Haverkort. Fitting World-Wide Web request traces with the EM-algorithm. *Performance Evaluation*, 52(2-3):175–191, 2003.

B.R. Haverkort, R. El Abdouni Khayari, and R. Sadre. A class-based least-recently used caching algorithm for World-Wide Web proxies. In *Computer Performance Evaluations, Modelling Techniques and Tools. 13th International Conference, TOOLS 2003*, volume 2794 of *Lecture Notes in Computer Science*, pages 273–290. Springer, 2003.

R. El Abdouni Khayari, R. Sadre, B.R. Haverkort, and A. Ost. The pseudo-self-similar traffic model: application and validation. *Performance Evaluation*, 56(1-4):3–22, 2004.

[3]In these papers the EM-fitting algorithm (without stratification) for hyper-exponential distributions is described. Sections 4.1 through 4.3 and parts of Section 4.6 are based on these.

Bibliography

[1] J. Abate and W. Whitt. Transient behavior of the M/M/1 queue: starting at the origin. *Queueing systems*, 2:41–65, 1987.

[2] N. Akar and K. Sohraby. An invariant subspace approach in M|G|1 and G|M|1 type Markov chains. *Communications in Statistics: Stochastic Models*, 13(3):251–257, 1997.

[3] D. Anick, D. Mitra, and M.M. Sondhi. Stochastic theory of a data-handling system with multiple sources. *Bell System Technical Journal*, 61(8):1871–1894, 1982.

[4] Apache Software Foundation. Apache HTTP Server Project. http://httpd.apache.org/.

[5] M.F. Arlitt and C.L. Williamson. Internet Web Servers: Workload Characterization and Performance Implications. *IEEE/ACM Transactions on Networking*, 5(5):631–645, 1997.

[6] S. Asmussen and O. Nerman. Fitting Phase-Type Distributions via the EM Algorithm. In *Symposium i Anvendt Statistik*, pages 335–346, 1991.

[7] S. Asmussen and R.Y. Rubinstein. Steady State Rare Event Simulation in Queueing Model and its Complexity Properties. *Advances in Queueing: Theory, Methods and Open Problems*, I:429–462, 1995.

[8] F. Baskett, K.M. Chandy, R.R. Muntz, and F. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM*, 22(2):248–260, 1975.

[9] N.G. Bean, D.A. Green, and P.G. Taylor. Approximations to the output process of MAP|PH|1 queues. In *Advances in Matrix Analytic Methods for Stochastic Models — Proceedings of the 2nd International Conference on Matrix Analytic Methods*, pages 151–159. Notable Publications, Inc., 1998.

[10] D. Bini, S. Chakravarthy, and B. Meini. A new algorithm for the design of finite capacity service units. In *Numerical Solution of Markov Chains (NSMC'99)*, pages 247–260. Prensas Universitarias de Zaragoza, 1999.

[11] D. Bini and B. Meini. On cyclic reduction applied to a class of Toeplitz-like matrices arising in queueing problems. In *Proceedings of the Second International Workshop on Numerical Solution of Markov Chains*, pages 21–38. Raleigh, North Carolina, 1995.

[12] A. Bobbio, A. Horváth, and M. Telek. Matching three moments with minimal acyclic phase type distributions. *Stochastic Models*, 21:303–326, 2005.

[13] P.P. Bocharov. Analysis of the queue length and the output flow in single server with finite waiting room and phase type distributions. *Problems of Control and Information Theory*, 16(3):211–222, 1987.

[14] P.P. Bocharov and V.A. Naoumov. Matrix-geometric stationary distribution for the PH/PH/1/r queue. *Elektronische Informationsverarbeitung und Kybernetik*, 22(4):179–186, 1986.

[15] G. Bolch, G. Fleischmann, and R. Schreppel. Ein funktionales Konzept zur Analyse von Warteschlangennetzen und Optimierung von Leistungsgrößen. In *Messung, Modellierung und Bewertung von Rechensystemen (MMB), Proceedings*, volume 154, pages 327–342. Springer, 1987.

[16] D. Brocker. Messung und Modellierung komplexer Verkehrsstrukturen in Hochgeschwindigkeitsnetzen. Diploma thesis, RWTH-Aachen, Germany, 1998.

[17] R. Brown. Calendar queues: a fast O(1) priority queue implementation for the simulation event set problem. *Communications of the ACM*, 31(10):1220–1227, october 1988.

[18] P.J. Burke. The output of a queueing system. *Operations Research*, 4:699–704, 1956.

[19] J.P. Buzen. Computational algorithms for closed queueing networks with exponential servers. *Communications of the ACM*, 16(9):527–531, 1973.

[20] R. Chakka. *Performance and Reliability Modelling of Computing Systems Using Spectral Expansion.* PhD thesis, University of Newcastle upon Tyne, 1995.

[21] S. Chakravarthy and M.F. Neuts. Algorithms for the design of finite-capacity service units. *Naval Research Logistics*, 36:147–165, 1989.

[22] G. Ciardo and A. Miner. SMART: Simulation and Markovian Analyzer for Reliability and Timing. In *Proceedings of the IEEE International Computer Performance and Dependability Symposium*, page 60. IEEE CS Press, September 1996.

[23] G. Clark, T. Courtney, D. Daly, D.D. Deavours, S. Derisavi, J.M. Doyle, W.H. Sanders, and P.G. Webster. The Möbius Modeling Tool. In *Proceedings of Petri Nets and Performance Models (PNPM 2001)*, pages 241–250. IEEE CS Press, Aachen, Germany, September 2001.

[24] M.E. Crovella and A. Bestavros. Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, 1997.

[25] J. Dahmen, K. Beulen, and H. Ney. A Mixture Density Based Approach for Object Recognition for Image Retrieval. 6*th International RIAO Conference on Content-Based Multimedia Information Access*, pages 1632–1647, 2000.

[26] D.D. Deavours and W.H. Sanders. Möbius: Framework and atomic models. In *Proceedings of Petri Nets and Performance Models (PNPM 2001)*, pages 251–260. IEEE CS Press, Aachen, Germany, September 2001.

[27] A. Dempster, N. Laird, and D. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society*, 39(B):1–38, 1977.

[28] R. El Abdouni Khayari. *Workload-Driven Design and Evaluation of Web-Based Systems*. PhD thesis, RWTH Aachen, Germany, 2003.

[29] R. El Abdouni Khayari, R. Sadre, and B.R. Haverkort. Fitting World-Wide Web request traces with the EM-algorithm. In *Proc. of SPIE 4523 (Internet Performance and Control of Network Systems)*, pages 211–220, 2001.

[30] R. El Abdouni Khayari, R. Sadre, and B.R. Haverkort. A validation of the pseudo self-similar traffic model. In *2002 International Conference on Dependable Systems and Networks (DSN 2002)*, pages 727–734. IEEE Computer Society, 2002.

[31] R. El Abdouni Khayari, R. Sadre, and B.R. Haverkort. Fitting World-Wide Web request traces with the EM-algorithm. *Performance Evaluation*, 52(2-3):175–191, 2003.

[32] R. El Abdouni Khayari, R. Sadre, B.R. Haverkort, and A. Ost. The pseudo-self-similar traffic model: application and validation. *Performance Evaluation*, 56(1-4):3–22, 2004.

[33] Y. Fang. Hyper-Erlang Distribution Model and its Application in Wireless Mobile Networks. *Wireless Networks*, 7:211–219, 2001.

[34] A. Feldmann and W. Whitt. Fitting Mixtures of Exponentials to Long-Tail Distributions to Analyze Network Performance Models. *Performance Evaluation*, 31:245–258, 1998.

[35] W. Fischer and K. Meier-Hellstern. The Markov-modulated Poisson process (MMPP) cookbook. *Performance Evaluation*, 18:149–171, 1992.

[36] G.S. Fishman. *Discrete-event simulation: modeling, programming and analysis*. Springer-Verlag New York Inc., 2001.

[37] B. Friis. Modelling Long-Range Dependent and Heavy-Tailed Phenomena by Matrix Analytic Methods. In G. Latouche and P. Taylor, editors, *Advances in Algorithmic Methods for Stochastic Models*, pages 265–278. Notable Publications, Inc., 2000.

[38] W.J. Gordon and G.J. Newell. Closed queueing systems with exponential servers. *Operations Research*, 15:254–265, 1967.

[39] C. Görg. *Rare Event Simulation. Simulation seltener Ereignisse und deren Anwendungen auf Kommunikationsnetze. Tutorial.* Lehrstuhl für Kommunikationsnetze, RWTH Aachen, September 1999.

[40] D. Green. Lag correlations of approximating departure processes for MAP|PH|1 queues. In G. Latouche and P. Taylor, editors, *Advances in Algorithmic Methods for Stochastic Models — Proceedings of the 3rd International Conference on Matrix Analytic Methods*, pages 135–151. Notable Publications, Inc., 2000.

[41] S.D. Gribble. UC Berkeley Home IP HTTP Traces.
http://www.acm.org/sigcomm/ITA/.

[42] M. Grossglauser and J.-C. Bolot. On the relevance of long-range dependence in network traffic. *IEEE/ACM Transactions on Networking*, 7(5):629–640, October 1999.

[43] B.R. Haverkort. Approximate analysis of networks of PH|PH|1|K queues: Theory & tool support. In H. Beilner and F. Bause, editors, *MMB*, volume 977 of *Lecture Notes in Computer Science*, pages 239–253. Springer, 1995.

[44] B.R. Haverkort. QNAUT: Approximately analyzing networks of PH|PH|1|K queues. *Proceedings of the 1996 International Computer Performance and Dependability Symposium*, page 57, 1996.

[45] B.R. Haverkort. Approximate analysis of networks of PH|PH|1|K queues with customer losses: Test results. *Annals of Operations Research*, 79:271–291, 1998.

[46] B.R. Haverkort. *Performance of Computer Communication Systems — A Model-Based Approach.* John Wiley & Sons, 1998.

[47] A. Heindl. *Traffic-based decomposition of general queueing networks with correlated input processes.* PhD thesis, Institut für Technische Informatik, Technische Universität Berlin, March 2001.

[48] A. Heindl. Decomposition of general queueing networks with MMPP inputs and customer losses. *Performance Evaluation*, 51(2-4):117–136, 2003.

[49] A. Heindl, Q. Zhang, and E. Smirni. ETAQA Truncation Models for the MAP/MAP/1 Departure Process. In *First International Conference on The Quantitative Evaluation of Systems (QEST'04)*, pages 100–109. IEEE Computer Society, 2004.

[50] H. Hermanns and J.-P. Katoen. Automated compositional Markov chain generation for a plain-old telephone system. *Science of Computer Programming*, 36(1):97–127, 2000.

[51] H. Hermanns and M. Siegle. Bisimulation Algorithms for Stochastic Process Algebras and Their BDD-Based Implementation. In J.-P. Katoen, editor, *Formal Methods for Real-Time and Probabilistic Systems, 5th International AMAST Workshop, ARTS'99, Bamberg, Germany*, volume 1601 of *Lecture Notes in Computer Science*, pages 244–264. Springer, 1999.

[52] A. Horvath and M. Telek. Approximating Heavy-Tailed Behaviour with Phase-Type Distributions. In G. Latouche and P. Taylor, editors, *Advances in Algorithmic Methods for Stochastic Models*, pages 191–213. Notable Publications, Inc., 2000.

[53] J.A. Incera Diéguez. *Contributions à la modélisation et à la simulation accélérée de réseaux de communication.* PhD thesis, Institut de Formation en Informatique et Communication, Université de Rennes, March 2001.

[54] J.R. Jackson. Networks of waiting lines. *Operations Research*, 5:518–521, 1957.

[55] R. Jain. *The Art of Computer System Performance Evaluation.* John Wiley & Sons, 1991.

[56] D. Janecek. Integrating FiFiQueues into the Möbius Framework. Diploma thesis, Lehr- und Forschungsgebiet Informatik 4, RWTH Aachen, 2004.

[57] M.A. Johnson. Selecting parameters of phase distributions: Combining nonlinear programming, heuristics, and Erlang distributions. *ORSA Journal on Computing*, 5(1):69–83, 1993.

[58] M.A. Johnson and M.R. Taaffe. Matching moments to phase distributions: Mixtures of Erlang distributions of common order. *Communications in Statistics: Stochastic Models*, 5(4):711–743, 1989.

[59] M.A. Johnson and M.R. Taaffe. Matching moments to phase distributions: Nonlinear programming approaches. *Communications in Statistics: Stochastic Models*, 6(2):259–281, 1990.

[60] P.J.B. King. *Computer and Communication Systems Performance Modelling.* Prentice-Hall, 1990.

[61] T. Koschel. Modellierung und Bewertung von Verteilten Web-Servern. Diploma thesis, Lehr- und Forschungsgebiet Informatik 4, RWTH Aachen, August 2002.

[62] D.D. Kouvatsos and N.P. Xenios. MEM for arbitrary queueing networks with multiple general servers and repetitive-service blocking. *Performance Evaluation*, 10:169–195, 1989.

[63] W. Krämer and M. Langenbach-Belz. Approximate Formulae for the Delay in the Queueing System GI/G/1. In *Proceedings of the 8th International Teletraffic Congress*, pages 235–1/8, 1976.

[64] P.J. Kühn. Approximate analysis of general queueing networks by decomposition. *IEEE Transactions on Communications*, 27(1):113–126, 1979.

[65] G. Latouche. Algorithms for infinite Markov chains with repeating columns. In C.D. Meyer, editor, *Linear algebra, Markov chains, and queueing models*, pages 231–265. Springer-Verlag, 1993.

[66] G. Latouche and V. Ramaswami. A logarithmic reduction algorithm for quasi birth and death processes. *Journal of Applied Probability*, 30:650–674, 1993.

[67] S.S. Lavenberg and M. Reiser. Stationary state probabilities at arrival instants for closed queueing networks with multiple types of customers. *Journal of Applied Probability*, 17(4):1048–1061, 1980.

[68] L.N. Le Ny and B. Sericola. Transient analysis of the BMAP/PH/1 queue. *International Journal of Simulation: Systems, Science & Technology. Special Issue on Analytical & Stochastic Modelling Techniques*, 3(3–4):4–14, December 2002.

[69] W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson. On the self-similar nature of Ethernet traffic. In *Proc. ACM SIGCOMM '93*, volume 23 of *Computer Communications Review*, pages 183–193, October 1993.

[70] P.S. Levy and S. Lemeshow. *Sampling of Populations: Methods and Applications*. Wiley, 1999.

[71] S.Q. Li. Queue response to input correlation functions: Continuous spectral analysis. *IEEE/ACM Transactions on Networking*, 1(6):678–692, December 1993.

[72] Y. Linde, A. Buzo, and R. Gray. An Algorithm for Vector Quantizer Design. *IEEE Transactions on Communications*, 28(1):84–95, 1980.

[73] M. Livny, B. Melamed, and A.K. Tsiolis. The impact of autocorrelation on queueing systems. *Management Science*, 39(3):322–339, March 1993.

[74] D.M. Lucantoni. New results on the single server queue with a batch Markovian arrival process. *Commun. Statist.- Stochastic Models*, 7(1):1–46, 1991.

[75] D.M. Lucantoni, G.L. Choudhury, and W. Whitt. Computing transient distributions in general single-server queues. In *Global Telecommunications Conference (GLOBECOM '93)*, pages 1045–1050. IEEE, 1993.

[76] D.M. Lucantoni, K.S. Meier-Hellstern, and M.F. Neuts. A single server queue with server vacations and a class of non-renewal arrival processes. *Advances in Applied Probability*, 22:676–705, 1990.

[77] A. Madansky. Optimal initial conditions for a simulation problem. *Operations Research*, 24:172–577, 1976.

[78] V. Mainkar and K.S. Trivedi. Sufficient conditions for existence of a fixed point in stochastic reward net-based iterative models. *IEEE Transactions of Software Engineering*, 22(9):640–653, September 1996.

[79] K.T. Marshall. Some inequalities in queueing. *Operations Research*, 16:651–665, 1968.

[80] M. Matsumoto and T. Nishimura. Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8:3–30, 1998.

[81] J. McKenna and D. Mitra. Asymptotic Expansions and Integral Representations of Moments of Queue Lengths in Closed Markovian Networks. *Journal of the ACM*, 31(2):346–360, 1984.

[82] V. Naoumov, U.R. Krieger, and D. Wagner. Analysis of a multi-server delay-loss system with a general Markovian arrival process. *Matrix-Analytical Methods in Stochastic Models*, 183:43–66, September 1996.

[83] Network Working Group. RFC 1945. Hypertext Transfer Protocol – HTTP/1.0. *http://www.w3.org/Protocols/rfc1945/rfc1945*, 1996.

[84] M.F. Neuts. A versatile Markovian point process. *Journal of Applied Probability*, 16(2):764–779, December 1979.

[85] M.F. Neuts. *Matrix-Geometric Solutions in Stochastic Models — An Algorithmic Approach*. Dover Publications, Inc., 1981.

[86] R.O. Onvural. Survey of closed queueing networks with blocking. *ACM Computing Surveys*, 22(2):83–121, june 1990.

[87] T. Osogami and M. Harchol-Balter. A Closed-Form Solution for Mapping General Distributions to Minimal PH Distributions. In Peter Kemper and William H. Sanders, editors, *Computer Performance Evaluation, Modelling Techniques and Tools. 13th International Conference, TOOLS 2003*, volume 2794 of *Lecture Notes in Computer Science*, pages 200–217. Springer, 2003.

[88] A. Ost. *Performance of Communication Systems – A Model-Based Approach with Matrix-Geometric Methods*. PhD thesis, Lehr- und Forschungsgebiet Informatik 4, RWTH Aachen, 2001.

[89] A. Ost and B.R. Haverkort. Analysis of windowing mechanisms with infinite-state stochastic Petri nets. *ACM Performance Evaluation Review*, 26(2):39–46, August 1998.

[90] A. Panchenko and A. Thümmler. Efficient phase-type fitting with aggregated traffic traces. *Performance Evaluation*, 2007. (to appear).

[91] A. Papoulis and S.U. Pillai. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, 2001.

[92] K. Pawlikowski. Steady-state simulation of queueing processes: A survey of problems and solutions. *ACM Computing Surveys*, 22(2):123–170, 1990.

[93] V. Paxson and S. Floid. Wide area traffic: The failure of Poisson modelling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995.

[94] P. Reinelt. Erweiterung des fixpunktbasierten Analyseverfahrens von FiFiQueues auf geschlossene Warteschlangennetze. Diploma thesis, Lehr- und Forschungsgebiet Informatik 4, RWTH Aachen, 2001.

[95] M. Reiser and S.S. Lavenberg. Mean-Value Analysis of Closed Multichain Queuing Networks. *Journal of the ACM*, 27(2):313–322, 1980.

[96] A. Riska, V. Diev, and E. Smirni. Efficient fitting of long-tailed data sets into hyperexponential distributions. *Internet Performance Symposium, IEEE GlobeCom 2002*, 3:2513–2517, 2002.

[97] A. Riska, V. Diev, and E. Smirni. An EM-based technique for approximating long-tailed data sets with PH distributions. *Performance Evaluation*, 55(1–2):147–164, 2004.

[98] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–29, March 1994.

[99] T. Ryden. Statistical Estimation for Markov-modulated Poisson Processes and Markovian Arrival Processes. In G. Latouche and P. Taylor, editors, *Advances in Algorithmic Methods for Stochastic Models*, pages 329–350. Notable Publications, Inc., 2000.

[100] R. Sadre and B.R. Haverkort. FiFiQueues: fixed-point analysis of queueing networks with finite-buffer stations. In *MMB (Kurzvorträge)*, volume 99-16, pages 77–80. Universität Trier, 1999.

[101] R. Sadre and B.R. Haverkort. FiFiQueues: fixed-point analysis of queueing networks with finite-buffer stations. In *Computer Performance Evaluation. Modelling Techniques and Tools: 11th International Conference, TOOLS 2000*, volume 1786 of *Lecture Notes in Computer Science*, pages 324–327. Springer, 2000.

[102] R. Sadre and B.R. Haverkort. Characterizing traffic streams in networks of MAP/MAP/1 queues. In *Proceedings 11th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB 2001)*, pages 195–208. VDE Verlag, 2001.

[103] R. Sadre, B.R. Haverkort, and A. Ost. An efficient and accurate decomposition method for open finite- and infinite-buffer queueing networks. In W. Stewart and B. Plateau, editors, *Proc. 3rd Int. Workshop on Numerical Solution of Markov Chains*, pages 1–20. Zaragosa University Press, 1999.

[104] H. Samelson. On the Brouwer Fixed Point Theorem. *Portugaliae mathematica*, 22(4):189–191, 1963.

[105] E.G. Schukat-Talamazzini. *Automatische Spracherkennung — Grundlagen, Statistische Modelle und Effiziente Algorithmen.* Friedrich Vieweg & Sons, 1995.

[106] P. Schweitzer. Approximate analysis of multichain closed queueing networks. In *Proceedings of the International Conference on Stochastic Control and Optimization*, 1979.

[107] K.C. Sevcik and I. Mitrani. The distribution of queueing network states at input and output instants. *Journal of the ACM*, 28(2):358–371, 1981.

[108] S. Söhnlein and A. Heindl. Analytic computation of end-to-end delays in queueing networks with batch Markovian arrival processes and phase-type service times. In *Proceedings of the 13th International Conference on Analytical and Stochastic Modelling Techniques and Applications (ASMTA), Bonn, Germany*, pages 1–7, May 2006.

[109] D. Tartemann. Untersuchung der Existenz eines Fixpunktes in einem iterativen Verfahren zur Warteschlangenanalyse. Diploma thesis, Lehr- und Forschungsgebiet Informatik 4, RWTH Aachen, 2002.

[110] A. Thümmler, P. Buchholz, and M. Telek. A Novel Approach for Phase-Type Fitting with the EM Algorithm. *IEEE Transactions on Dependable and Secure Computing*, 3:245–258, 2005.

[111] A. Thümmler, P. Buchholz, and M. Telek. A novel approach for fitting probability distributions to real trace data with the EM algorithm. In *International Conference on Dependable Systems and Networks, 2005. DSN 2005. Proceedings*, pages 712–721. IEEE CS Press 2005, 2005.

[112] M. Villén-Altamirano and J. Villén-Altamirano. RESTART: a straightforward method for fast simulation of rare events. In *Proceedings of the 1994 Winter Simulation Conference*, pages 282–289, 1994.

[113] A.J. Weerstra. Using matrix-geometric methods to enhance the QNA method for solving large queueing networks. Diploma thesis, Department of Computer Science, University of Twente, 1994.

[114] W. Whitt. Approximating a point process by a renewal process, I: Two basic methods. *Operations Research*, 30(1):115–138, 1982.

[115] W. Whitt. The Queueing Network Analyzer. *The Bell System Technical Journal*, 62(9):2779–2815, 1983.

[116] W. Whitt. Performance of The Queueing Network Analyzer. *The Bell System Technical Journal*, 62(9):2817–2843, 1983.

[117] J.L. Zahorjan, K.C. Sevcik, D.L. Eager, and B.I. Galler. Balanced job bound analysis of queueing networks. *Communications of the ACM*, 25(2):134–141, 1982.

Index