Efficient System Evaluation Using Stochastic Models

Philipp Reinecke

Dissertation

Eingereicht beim Fachbereich Mathematik und
Informatik der Freien Universität Berlin.

Betreuerin: Prof. Dr. Katinka Wolter
Erstgutachterin: Prof. Dr. Katinka Wolter
Zweitgutachter: Prof. Dr. Miklós Telek
Datum der Disputation: 12. Oktober 2012

2012

Summary

Efficient and accurate methods are indispensible in the evaluation of complex systems. Methods of various abstraction levels may be applied, ranging from measurements in test-beds over simulation to analytical approaches. These methods vary in their efficiency and accuracy. In order to obtain realistic results, methods from all abstraction levels should be applied. Furthermore, real-world phenomena must be represented in a suitable way at all abstraction levels. This approach requires the use of stochastic models for the faults that may affect a system. This thesis explores several important issues of system evaluation using stochastic fault models.

The first part of the thesis provides a survey of stochastic fault-models for system evaluation. We survey a broad range of existing fault-models and provide a new classification scheme for fault models in service-oriented systems.

In the second part of the thesis we address the use of phase-type (PH) distributions as fault models in system evaluation. We study algorithms for random-variate generation and discuss their computational costs. An optimisation method for reducing the cost of random-variate generation by transformation of the representation is presented and an optimality result for the sub-class of acyclic phase-type distributions is derived. We then develop a new algorithm for efficient and user-friendly fitting of PH distributions to data sets using clustering. The algorithm is implemented as part of the HyperStar tool. We additionally illustrate application of HyperStar as a graphical user interface for prototype algorithms using a new algorithm for fitting general PH distributions.

The third part illustrates application of the results from the first and second parts. We first study the application of stochastic fault models for IP packet loss in fault-injection and investigate the interaction between fault models and traffic arrival streams and its impact on evaluation results. We then describe a library for random-variate generation from phase-type distributions and show its use in two case-studies.

'It is the nature of an hypothesis, when once a man has conceived it, that it assimilates everything to itself, as proper nourishment; and from the first moment of your begetting it, it generally grows the stronger by everything you see, hear, read, or understand. This is of great use.'

Laurence Sterne, The Life and Opinions of Tristram Shandy, Gentleman, p.104f [Ste69]

Contents

Introduction

System evaluation is an indispensible part of the design and development of computer and communication systems. It provides us with insights into important performance, dependability, and security properties of the systems we study, thus fostering an understanding of the status quo as well as providing directions for future developments.

As a running example for the kind of questions asked in system evaluation, let us consider the problem of computing the restart timeout in a distributed client-server system. The client uses this timeout to detect if a request takes very long to complete or has failed altogether. When the timeout elapses, the client aborts and restarts the request. If failures and long completion-times are caused by transient faults, the restarted request may receive a response that arrives sooner than if the client had waited for the response to the original request. Measurement studies show that large response-times or failed requests can occur due to transient faults such as IP packet loss and temporary server overload [RvMW04, RvMW06a, RW08b, RWW09]. In many scenarios, the restart method can be used to improve both performance and dependability. However, the restart timeout must be adapted to the scenario. If the timeout is too small, the client may restart too often, thereby increasing the load and exacerbating the problem. If, on the other hand, the timeout is too large, restart will have litte effect on the response-times, as it is very unlikely to be triggered. In order to develop algorithms for computing the restart timeout we first need to understand the behaviour of existing systems in terms of response-times and request failures. We must then quantify the effects of restart, in order to determine if our algorithms reduce response-times and the number of failed requests.

Such problems can – and ideally should – be studied at different abstraction levels. Experimentation on test-beds is necessary to gain an understanding of the object of study under realistic conditions. On the other hand, formalisation of the problem within an abstract mathematical framework is a prerequisite for obtaining general results. Abstraction levels generally differ in the type of methods that are available: At a high abstraction level, analytical methods can be applied, while more detailed system models may become analytically intractable and require simulation. Finally, the evaluation of realistic system behaviour under controlled conditions re-

quires experimentation using system test-beds. Typically, the higher the abstraction level, the more elegant the solution methods. In particular, this affects the speed with which solutions can be obtained, and thus the range of parameter variations that can be explored. On the other hand, the attractiveness of elegant solution techniques and general results comes at the disadvantage that a high-level abstraction may abstract away important aspects of the problem (or even the problem itself). Combining approaches allows us to cross-check results obtained at different abstraction levels and thus ensures that our results are both meaningful and general.

Referring to our running example, the question how to best compute the timeout may be addressed using several approaches at different abstraction levels. One may start by comparing implementations of restart algorithms using a test-bed with fault-injection [RvMW06a, RW08a]. However, the results are then specific to the scenario reflected in the test-bed, and experiments often require a considerable amount of time. Insights that apply to other scenarios require more abstract approaches. For instance, based on their model of response-times as a random variable the authors of [vMW06] develop an algorithm for computing the restart timeout that minimises response-times. This algorithm is very general in that it can be applied to all situations where one is interested in reducing response-times. On the other hand, it is limited in that the underlying model only considers restart by a single client. The common situation where several clients apply restart requires a less abstract model, such as the queueing-models proposed in [WR08, WR10]. Unfortunately, in contrast to the one in [vMW06], these models cannot be solved analytically and require simulation, which renders results less general. Furthermore, simulation often requires more time than the analytical solution of [vMW06].

Irrespective of the abstraction level, it is helpful to think of a system evaluation as being performed on a *system model*, which models the relevant behaviour of the system. When studying problems of performance, dependability, or security we are especially interested in the behaviour of the system under various common faults or perturbations. System behaviour is assessed using *measures* defined for the system model. For instance, a typical measure for queueing-network models is the job completion time, while a common measure in a testbed is response-time.

In order to use a system model to study the effect of faults or perturbations on a system, we must be able to add models for these faults to the system model. Recurring to well-known terminology from fault-tolerant computing, we call these models for faults *fault models*, but extend the classical definition by requiring that the model allows us to reproduce the fault in a realistic way. Fault models are parameters to the system model. Such a clear distinction between the fault model and the system model is common in fault-injection experiments for dependability benchmarking, where one explicitly describes a fault-load that the system is subjected to [vMBP+08, VM02]. We apply this concept to system models in general,

by describing the factors that affect the measures by fault models, which we treat as parameters to our system model. Such parameterisation may be as simple as setting a service rate in a queueing-network model, or a constant delay in a testbed. In this case, the parameter to the system model is the constant service rate or the constant delay. However, service rates or delays may vary over time. If we have a model for these variations, we can replace the constants by this model. In doing so, we set the fault-model parameter of the system model to be the model expressing service-rate or delay variations.

This notion of fault models as parameters to system models implies that we can obtain the same measures with the same interpretation from the system model, regardless of the fault models that are used to represent faults. That is, using the terminology of functional and non- (or extra-) functional behaviour, we require that the system model maintains the same functional behaviour and the same system structure when we change the fault model. This clarification is necessary because with some model classes the system model may change significantly when the fault model is changed. For instance, introducing response-times modelled by phase-type (PH) distributions into a Continuous-Time Markov Chain (CTMC) model will lead to a drastically different CTMC, as this is accomplished by replacing a single transition by the Markov chain underlying the PH distribution. Still, the same measures, such as first-passage time, can be computed for the original and the modified model, and have the same interpretation for both.

The distinction between system models and fault models highlights a few requirements often encountered in system evaluation. First, fault models used in the evaluation must of course accurately reflect properties of the real world. Second, it should be possible to use the fault models in different types of system models, such that methods from the different abstraction levels can be used. Third, the resulting models should allow efficient evaluation methods. Fourth, fault models should be such that they can easily be derived and used, and, finally, fault models for typical faults must be availabe.

Contributions

The contributions in this thesis help to address these requirements. The first part of the thesis surveys fault models that are available for system evaluation and studies issues of their application. The second part provides several improvements to the state of the art in using phase-type distributions as fault models. The third part illustrates these concepts in several applications.

In the following we give a short overview of these contributions. A chapter-by-chapter summary is provided in the section titled *Organisation of the Thesis*, below.

Part I: Stochastic Fault Models for System Evaluation In the first part of the thesis we consider stochastic fault-models for system evaluation in general. Irrespective of the method used when evaluating a system, an important problem is the need for realistic models of the faults that can occur. Without realistic fault models, evaluation results cannot be expected to be relevant. As discussed in the second part, phase-type distributions are one important tool in modelling faults, but many other types of models do exist. Ideally, fault-models are derived from measurements of the real system. In practice, however, the real system may not be accessible to the researcher or it may not even exist yet. These difficulties are particularly relevant for complex distributed systems consisting of many layers of complex components. With these systems, models that can reproduce the fault-behaviour of the component systems accurately are required in analytical, simulation, and experimental studies. In this work we discuss existing fault-models for component systems and investigate the importance of accurate fault-models for IP packet loss in simulation studies.

There exists a large number of measurement studies of various systems that are often components in larger systems (e.g. the network or a web server), and in recent years many measurement traces have been made available in data repositories (e.g. the AMBER data repository [AMM10]). Models for the failures observed in these studies can be used as fault models in larger systems, but existing results are scattered widely, and for many component systems no fault models exist.

Using a service-oriented system as an example, we survey existing fault models and classify them according to previously-proposed fault classification schemes. Our survey gives an overview of fault models that allows the reader to select the required fault models for system evaluation. With these fault models, the system model can be parameterised such that it reflects the properties of a real-world system. Additionally, we provide a fault-classification scheme that is well-suited to classifying fault models. Our survey also helps to identify areas where no suitable fault models are available yet.

The work in this part helps the reader to quickly and accurately parameterise system models for evaluation studies using testbeds, simulation, and analytical approaches, such that the studies provide realistic results.

Part II: Phase-type Distributions The second part of the thesis focusses on phase-type distributions, which can be used as a special type of fault-model. Phase-type (PH) distributions [Neu81] are a well-known tool in the analytical community, but are seldom used in simulation or experiments. For example, the well-known simulation-tools NS-2 [ns2], OMNet++ [Var01] and Möbius [DCC+02] do not support phase-type distributions as part of their set of standard distributions, and the same holds for system test-beds, e.g. the SOA test-bed generator GENESIS [JTD08].

While in each of these tools PH distributions may be explicitly modelled, they cannot be used as a ready-made module, in the same way that other distributions are supported. As PH distributions can fit many properties of empirical distributions very well and are also very well suited for analytical approaches, their application as fault models in simulation would fill a gap between the abstraction levels, as it helps to use the same fault models throughout all levels. This work focusses on random-variate generation and fitting of PH distributions as two important aspects of the use of these distributions.

Random-variate generation from PH distributions has received very little attention, both with respect to the development and implementation of algorithms and with respect to the performance costs involved. In 1981, Neuts and Pagano [NP81] proposed a first algorithm with optimisations for more efficient random-variate generation. This algorithm operates on the underlying structure of a PH distribution. Brown et al. [BPdL98] explored numerical inversion of the distribution function as a way of generating Phase-type and Matrix-Exponential random variates. Recently, Horváth and Telek [HT11] developed an algorithm that uses the acceptance/rejection technique to improve efficiency. Kriege and Buchholz [KB11] support PH distributions as part of their module for generating stochastic processes in OMNeT++, and thus provide tool support for the PH class. They implement a straightforward algorithm without any optimisations for efficiency.

This work addresses random-variate generation from PH distributions both from a theoretical and a practical point of view. We survey existing approaches and present two new algorithms. Complexity of the random-variate generation algorithms is compared in terms of atomic operations. We give a result for the optimal representation of Acyclic Phase-type distributions in terms of the computational costs of random-variate generation. Although the result does not extend to the whole PH class, optimisation is still possible for general PH distributions, and we provide heuristic algorithms for the optimisation process.

In contrast to simulation, fitting of PH distributions to data sets is a well-explored area. Approaches include moment-fitting (e.g. [BPdL98, TH02a, BHT05, CZS08]), Expectation-Maximisation (EM) algorithms (e.g. [ANO96, RDS02, TBT06, WLS08, SH08]) and methods from non-linear optimisation (e.g. [HT02]). These approaches differ in the PH classes they support and in the fitting quality they provide. From a practical point of view, tool support for PH fitting is equally important as the fitting quality. Although not all of these methods are implemented in ready-to-use tools, there exist several implementations: Moment-fitting is implemented in e.g. the KPC-Toolbox [CZS08], EM algorithms are employed in e.g. G-FIT [TBT06] and EMPHT [ANO96], and non-linear optimisation is used in PhFit [HT02].

This work augments the set of tools for PH fitting in two ways. First, a cluster-based approach is proposed that allows efficient identification and fitting of important features of the empirical data, such as peaks in the density. The method lends

itself to intuitive optimisation of the fitting quality, as the user can set initial parameters by graphical selection of important properties. The algorithm forms the basis of the HyperStar tool, which is an extensible tool for PH fitting. HyperStar produces PH distributions that capture the density of typical distributions better than established tools and are well-suited for simulation at the same time. Second, the modular architecture of HyperStar allows its use as a graphical user interface for new fitting algorithms. This is illustrated in a case-study of a new algorithm for fitting general PH distributions. The algorithm gives results comparable to existing tools, but provides sparse representations.

The work presented in the second part enables efficient and user-friendly application of the PH class throughout all abstraction levels in system evaluation. This allows the use of an integrated evaluation approach using analysis, simulation and experimentation, thereby improving the quality and significance of system-evaluation results.

Part III: Applications The third part illustrates these concepts using several case-studies. We first discuss the choice of appropriate fault-models for IP packet loss, before illustrating the application phase-type distributions in efficient simulation.

IP packet loss models are an important instance of fault models due to the prevalence of networked systems. In particular, loss models are required to reproduce realistic working conditions in testbed-based evaluations of networked systems. Gilbert models provide a good description of packet-loss dynamics in the Internet (e.g. [ZPS00, ZDPS01, HGHl08]), since they can represent alternating phases of high and low loss levels. Gilbert models describe the packet-loss process by a Markov chain with different loss levels for the states of the chain. Such models are commonly derived from traces of the packet loss between two hosts. A Gilbert model can be understood both as an embedded Discrete-Time Markov Chain (DTMC) and as a Continuous-Time Markov Chain (CTMC), and both models can describe packet-loss traces equivalently well. We study the use of both interpretations of Gilbert models for packet-loss injection in a test-bed, and show that models which are identical from the point of view of *describing* the packet-loss process can produce significantly different results when *reproducing* the process. The Linux kernel module NetemCG, which was developed as part of the evaluation, enables the injection of packet loss according to a continuous-time Gilbert model. Results obtained using this model appear more realistic than those of a discrete-time Gilbert model, as implemented in [SLO10].

Our findings for efficient random-variate generation from phase-type distributions are implemented in the libphprng library, which is part of Butools [BBH$^+$11]. With these contributions, efficient simulation of PH distributions becomes applicable in a wide range of scenarios, as the library can easily be used in popular

tools, such as OMNeT++ and NS-2. As illustrated in a case-study, the efficient algorithms developed here enable simulation speeds almost four times as fast as the approach in [KB11]. A second case-study demonstrates the performance gain that can be achieved by using PH distributions to approximate complex sub-models in a simulation.

Organisation of the Thesis

In the first part of the thesis we take a general perspective on fault models. Part I has the following structure:

Chapter 1 surveys existing fault models using a recursive approach derived from the structure of a service-oriented system. We discuss fault-classification schemes for service-oriented systems and illustrate their application to order the fault models. We identify areas where the behaviour of faults is still not well-understood and discuss shortcomings of the existing classification schemes. These shortcomings are addressed by the classification scheme that we use throughout the survey. An earlier version of the work in this chapter is available in [RWM10].

The second part of the thesis concerns various aspects of using phase-type distributions as fault-models in system evaluation. Part II is structured as follows:

Chapter 2 summarises existing work on phase-type distributions in order to provide the necessary mathematical background. This chapter introduces basic notation, special and canonical representations, important properties, and operations used throughout the remaining chapters of the first part of this work.

Chapter 3 discusses algorithms for random-variate generation from PH distributions. The chapter gives an overview of existing algorithms before embarking on a detailed study of algorithms that rely on playing the Markov Chain of the PH distribution. We show that the structure of the representation affects efficiency of random-variate generation. We identify metrics for the cost of random-variate generation and derive expressions for these metrics that are based on atomic operations. A measurement study illustrates that the costs of different atomic operations may differ significantly. This chapter is based on work which has been presented before in [RWBT09, RTW10, HRTW12, RBD12].

Chapter 4 describes optimisation of phase-type distributions for efficient random-variate generation. Starting from the canonical form for the Acyclic Phase-type (APH) class, we derive a theoretical result on the optimal representation for any

APH distribution. We develop two algorithms that compute the optimal representation. We show that the optimality result does not extend to the whole PH class and define heuristic algorithms that help to reduce the computational costs. An empirical study explores typical cost savings available using the proposed approach. The contributions in this chapter are based on the work published in [RTW10, HRTW12].

Chapter 5 discusses fitting of phase-type distributions to data. The emphasis in this chapter is on user-friendly fitting approaches that provide distributions well-suited for random-variate generation using the methods introduced in Chapters 3 and 4. We devise a clustering-based fitting algorithm that yields good fitting results with little user interaction. The approach is implemented in the HyperStar tool. HyperStar is extensible and can also act as a graphical user-interface for prototypes of new fitting methods. This use of the tool is illustrated in the evaluation of a new algorithm for fitting general phase-type distributions with sparse representations. The cluster-based fitting algorithm and the HyperStar tool have been presented in [RKW12a, RKW12b].

The third part of the thesis illustrates application of the results of the previous parts and is structured as follows:

Chapter 6 discusses packet-loss injection as a means of emulating IP packet loss in experiments on test-beds. We consider Bernoulli loss and two variants of Gilbert models and measure their effect on Ping and TCP arrival streams. We compare an implementation of an embedded DTCM Gilbert model to our own implementation of a continuous-time Gilbert model. Our observations show that with Ping arrival streams all models produce the same result, but that the embedded DTMC models result in extended outage periods with TCP. The work in this chapter is based on [RDW11, RW11].

Chapter 7 applies the findings for phase-type distributions in two case studies. We introduce the Libphprng library for efficient random-variate generation from PH distributions. We illustrate its application and evaluate its performance in a case-study on the optimal restart interval. This case study further shows the importance of choosing realistic fault models for obtaining realistic results. The second case-study shows the efficiency improvements that can be obtained by applying phase-type distributions to approximate the behaviour of highly-detailed component models in simulation studies. The contributions in this part are also available in [WRM11, RH12].

The main part of the thesis is concluded with a summary and an outlook on future work. The appendices contain details on several auxilliary aspects, such as small proofs, and summarise some theoretical basics used in this work.

Acknowledgments

Many people were influential to the writing of this thesis and deserve my deepest gratitude. First and foremost, I want to thank my family for their unwavering support of my various academic endeavours. Second, I want to thank Katinka Wolter and Miklós Telek for their effort in supervising and reviewing this thesis, and for their many helpful comments throughout the process. Third, I would like to thank Levente Bodrog, Alexandra Danilkina, Gábor Horváth, Tilman Krauß, Miroslaw Malek, Alfons Mittermaier, Miklós Telek, and Katinka Wolter, who were involved as co-authors in some of the work presented here. Finally, I want to thank the colleagues and students in the groups at Humboldt-Universität zu Berlin and Freie Universität Berlin for providing an inspiring work environment.

Thank you.

Part I

Stochastic Fault-Models for System Evaluation

Chapter One

A Survey on Fault Models for QoS Evaluation of Complex Distributed Systems

System evaluation requires realistic fault models to deliver relevant results. Recurring to our example of the evaluation of restart algorithms, in order to obtain useful results from an evaluation in either a testbed, a simulation, or an analysis, we need realistic models of the faults behaviour in systems where restart is to be applied. Such models may be obtained through measurement studies by the researcher prior to the actual evaluation, or they may be taken from published work that focusses on measurement studies. The latter approach has the advantage that it saves time and that it enables the use of models for systems to which we have no access. Unfortunately, although a lot of measurement studies have been published, the area is quite fragmented, with little to no general overviews of models that have been obtained. Furthermore, the derivation of fault models is often not the primary concern of measurement studies, and therefore not many models of the measurements are available.

In this chapter we provide an overview of existing fault-models that can be used in system evaluation using various approaches. In particular, we are interested in fault-models for complex distributed systems, and we use service-oriented systems as our example application. Service-orientation is an important paradigm for today's computing systems. A service-oriented system[1] consists of many individual services, often provided by independent business entities, that communicate over the Internet in order to provide a composed service. With their increasing success in the business environment, Quality of Service (QoS) issues of service-oriented systems

[1]We employ the terms service-oriented system and system with service-oriented architecture (SOA system) synonymously.

have become important. Our survey offers two major contributions to the use of stochastic models in system evaluation, particularly with respect to service-oriented systems: First, we provide an overview of the fault models available in the literature, using a fault-classification scheme based on architectural properties of a SOA system. Second, we review schemes that have been proposed for classifying faults in SOA systems.

The first part (Section 1.2) aims to give the researcher interested in QoS issues a structured overview of available fault models that may be used for parameterisation. It also highlights which components of SOA systems are as of yet lacking descriptions suitable as fault models, and may thus serve to direct measurement efforts to these components. The survey of fault models complements data repositories such as the AMBER data repository [vMBP+08, vMAB+09, AMM10] in that we summarise models and insights, whereas data repositories gather measurement data for further analysis.

In the second part of the chapter we give an overview of fault-classification schemes for SOA systems and apply them to the fault models surveyed in the study. Our review is intended to serve as a guideline for a structured approach to building and parameterising system models. The classification schemes enable researchers to develop a system model that reflects the aspects of the service-oriented systems most relevant to their work. Using our classifications of fault models, they may then select fault models to reproduce faults of interest and study their effects. We anticipate that both parts will help to improve the quality of analytic, simulation, and experimentation-based studies of service-oriented systems by enabling the development of system models that are not limited to one specific system, but rather reflect general properties.

Note that while our discussion focusses on service-oriented systems for reasons of clarity, the fault models we identify can be applied in studies of many other distributed computing systems. We chose to focus on service-oriented systems in order to have a distinct definition of the system for which we require fault models. SOA systems are typical for many distributed systems, in that they are highly-distributed, tremendously complex, and often opaque to the user. We illustrate the application of our survey to another type of distributed systems in Section 1.4, where we discuss performance evaluation of Cloud Computing.

1.1 Preliminaries

In this section we first provide a definition of the concept of a fault-model, as we employ it in the following. In order to clearly convey the underlying idea, we contrast it to existing definitions from fault-tolerant computing. We then describe the structure of the survey from a methodological point of view, as this aids the understanding and application of the survey.

1.1.1 What Constitutes a Fault Model?

In dependable and fault-tolerant computing systems, fault models have long been used to describe certain fault-classes such as crash, omission, timing, or Byzantine faults [LSP82, Cri91, RLT78, ALRL04]. These classic models have proved useful in the development of algorithms that make systems tolerant towards faults of a certain class. On the other hand, classic fault models are not sufficient for the quantitative evaluation of the effects of faults on quality of service, because they focus on the qualitative description of the fault and do not encompass occurrence characteristics. In particular, temporal patterns are not captured by classic fault models. That is, while a classic fault model tells us the type of fault that can occur (e.g. omission of a message), it does not convey information on when the fault occurs or about the likelihood of fault occurrence. Consequently, classic fault models do not allow realistic replication of faults under controlled conditions.

As described above, we view fault models as parameters to a system model that influence the modelled system's QoS. Our definition of fault models suitable in this context can be derived from the basic concepts of fault-tolerant computing. Following [Cri91], we employ the notion of systems (servers) that implement a set of functionality (the service) to be used by other systems (users, clients), which in turn may be servers (and thus provide a service) to further systems. The distinction between server, service and client is generic enough to be made in system models in general, although this may not always be as obvious as with e.g. a queueing-network model, where jobs are the clients, server stations are the servers, and the service is provided by transition through the server stations. The service provided by a server is *correct* if it is in accordance with the specification for the service, which may define both functional and non-functional properties. Based on [ALRL04], we call deviation from the service specification a *service failure*, and the cause of a failure a *fault*.

Now, as the server provides its service to a client, which again can be thought of as a server to other clients, a service failure in the server may be the cause of a failure of the service the client in turn provides to its own clients, and hence the server's failure is a fault for the client. Different failures of the server (i.e. different deviations from correct service) constitute different faults to the client and may result in different client failures. Our notion of fault models as parameters to a system model follows from this observation: By changing the characteristics of the deviations from correct service we can induce different faults in the client system. Consequently, to model the faults that may affect a system we need a description of the failures of the server(s) that the system under study depends on. *A fault model, in a general sense, should therefore consist of descriptions of the server, the service, and the characteristics of the deviations from correct service that constitute a failure.* Note that the latter often depend on the service as well. With response times, for

instance, there may be a threshold beyond which service responses become useless to the client; this threshold, however, may vary between clients. To account for this, we relax our definition such that we also include models that do not explicitly provide a description of the deviations from correct service that amount to a failure, if they enable the derivation of such a description.

In the following, we assume that failure location is given by the description of the server. That is, given a server providing a service, we focus on the characteristics of the deviation from correct service. Based on the different ways to describe deviation characteristics, we divide the large set of fault models that fit the above definition into the following three types, based on how the models describe failure occurence characteristics (in the server system) or, equivalently, fault occurence characteristics (in the client system):

Bernoulli Trials With these, the fault is described as a Bernoulli-distributed random variable. That is, the occurrence of the fault is defined by a probability p. Typical examples for this type are fault models that reflect service availability/unavailability, e.g. a service is unavailable with a probability of $p = 0.05$. Bernoulli trials may be used to describe the case where no faults occur ($p = 0$) as well as for deterministic faults ($p = 1$). Note that Bernoulli trials do not encompass a notion of time.

General Random Variables Here, fault occurrence is described by a random variable with a distribution that is more general than the Bernoulli distribution. Ideally, the distribution should be specified completely, e.g. by its distribution function, but we will also include less complete characterisations, e.g. a mean or a quantile of the distribution. Examples for this kind of fault model are response-time distributions. As different operating conditions may affect the distribution, more elaborate models of this category allow parameterisation. For instance, response-times may vary with system load, and this may be expressed by individual response-time distributions for the different load-levels.

Stochastic Processes These provide a characterisation of faults over time, capturing more complex random structures, e.g. correlation. We use a very loose definition of a stochastic process, in that we only require that the fault model contains a description of states and of state transitions. The states may provide arbitrarily complex characterisations of a fault, while the state transitions serve to define the time-dependent behaviour. For instance, the classic Gilbert-model (cf. e.g. [HGHl08]), which describes a no-loss and a loss state and transition probabilities between both, is a stochastic process. Note, however, that we do not put any restrictions on the distributions characterising state-transitions. In particular, we do not limit the focus to Markovian models.

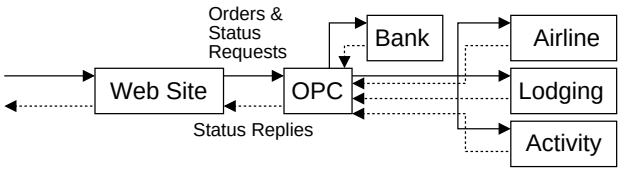


Figure 1.1: Functional structure of the SUN Java Adventure Builder reference implementation.

Let us illustrate our definitions using a simple example: In [RW08b] we presented Acyclic Phase-type (ACPH) models of the transmission delays experienced when sending SOAP messages over a network link with IP packet loss. Messages were transmitted over WSRM (Web Services Reliable Messaging, [BIMT05]), which provides a reliable message transport within the SOAP stack. Furthermore, the SOAP transport used by the WSRM layer was HTTP over TCP, the latter of which provides fault tolerance for IP packet loss.

In this case, the server is a link capable of reliably transmitting SOAP messages, the service is end-to-end message transmission, and the client is an application that depends on timely SOAP message transmission. We want to use this fault model in a queueing network model of a SOA system. The definitions of the server, service and client are required to determine where we can use this fault model in the SOA system model. E.g., we can obviously not use it for a Web Service, but we can use it for the network link required to invoke the Web Service. The behaviour of the service of the link is described by ACPHs, which are general random variables according to our above distinction. Based on these, we can define when a fault occurs by e.g. defining a threshold consistent with the requirements of the application. However, it may be preferable to use the distributions directly, thus leaving the exact definition of a fault implicit, and only observing the effects of faults on the application.

1.1.2 Structure of the Survey

Before embarking on our survey of available fault models, we need to define a structured approach to the review of the literature. Fault-classification schemes help to obtain a comprehensive overview, since they describe which faults we may observe in a SOA system, and hence have to find models for. In Section 1.3 five fault-classification schemes for service-oriented systems are discussed. Unfortunately, as illustrated there, these schemes do not provide clear-cut criteria for many common faults. This disadvantage manifests in overlaps between fault categories. Furthermore, the schemes tend to consider only abstract fault classes, and thus the guidelines they provide are incomplete.

For these reasons, we employ a recursive system-based approach: We start with the typical structure of a system and try to find fault models for the behaviour of each of the components of the system at a high abstraction-level. We then look at the typical structure of each of the components and repeat this recursion until we have obtained suitable fault models or until further recursion becomes unlikely to yield further insight.

We can derive the typical structure of a SOA system from the SOA reference application SUN Java Adventure Builder [Sun06], which we briefly describe in the following. The system implements an online travel agency that users can interact with using a web site, accessible via HTTP over the Internet. The web site back-end (Order Processing Center, OPC) calls four Web Services (WS), using the SOAP protocol, which again translates to HTTP interactions over the Internet. Each of the services (Bank, Airline, Lodging, Activity) shown in Figure 1.1, as well as the web-site and back-end (OPC), corresponds to a physical system implementing the functionality of the provided service. In the case of the web site, this system typically consists of an HTTP server running on top of an operating system, which in turn depends on the underlying hardware. The Web Services are deployed within application servers, which again require an operating system (and hardware). Frequently, services rely on storage systems such as databases for their operation. Furthermore, the hardware beneath the operating system does not need to be a physical machine, but may instead be a virtual machine that shares the physical hardware with several other virtual machines. Faults in each of these (and their sub-systems) may affect quality of service experienced by the user. A closer look at the communication links between the components shows that a similar dissection can be performed here. Faults can occur in the Web Services stack and in the Internet underlying the WS stack. In both cases, the infrastructure and the transport itself are of interest.

This recursive approach to structure our survey extends the number of potential sources for fault models by avoiding the limitation to studies specific to SOA systems. For instance, Internet behaviour has been studied independently, and fault models derived there are useful in SOA systems, which rely on message transmissions. On the other hand, the structure implied by the recursion helps to limit the number of studies to be considered. E.g. CPU register faults affect QoS of a SOA system only if they cause faults of the platform, and thus they are not of interest in themselves.

1.2 A Survey of Available Fault Models and Data

This section presents a survey of fault models available in the literature. We start with individual Web Services and typical sub-systems they rely on. We then consider the communication part, using the same method. Our recursive approach results in the following structure:

1. Web Service
    a) Platform
        i. Application Server/SOAP Framework
        ii. Virtualisation
    b) Storage/Databases
2. Communication
    a) Web Services Stack
        i. Infrastructure
        ii. Message Transport
    b) Internet
        i. Internet Infrastructure
        ii. Transport

In terms of the definition of a fault model, each item in this list describes a server that may deviate from its intended service. Thus in each of the following subsections we consider studies that describe, or at least allow us to derive, characterisations of the deviations from correct service for the respective server. Descending through the layers listed above, at each layer we look for Bernoulli, Random-Variable and Stochastic Process models of faults at the respective layer. For instance, at the Service level a Bernoulli model would describe availability of the service to requests, while a Random-Variable model might describe the times between failures or the response-times of the service. A Stochastic-Process model, in turn, could even encompass variations in response-times over subsequent requests. Similarly, if we look at the Internet Transport layer, a Bernoulli model could be used to describe uniform packet loss, a Random-Variable model could reflect delay characteristics, and a Stochastic-Process model could describe variations in loss or delay.

The objective of this section is to provide a structured overview that lists suitable fault models of each class at each layer. Table 1.1 on page 21 summarises the surveyed fault model in this way.

1.2.1 Web Service

At the highest level, we study a system consisting of a Web Service and a user (human or machine), e.g. in Figure 1.1 the back-end (Order Processing Center, OPC) is a Web Service that is accessed by the Web Site. In our model of this system, the Web Service is the server whose failures constitute the faults that affect the client. Thus we require fault models that reflect the behaviour of Web Services, as seen by their

users. With our focus on quality-of-service aspects, we are interested in faults that manifest as either unavailability of the service (i.e. the service does not respond to or rejects requests) or insufficient speed of responses.

Bernoulli models for the availability of Web Services can be derived from the study of the availability of Web Services in a publicly-accessible service repository in [KR04]. The paper states that the availability of services in the repository is 0.84. More recent studies are available in [SF07, ZL08, ZZL10]. [SF07] provides availability values for different public Web Services, including Amazon WS and Google WS. The study shows that availability may vary between 0.82 for the Google Web Service and 1 for several Amazon Web Services. In [ZL08], Zheng et al. observe Amazon Web Services in several different locations and two public Web Services in the USA using accesses from different countries. Unavailability values between 0 and 0.82 are reported, depending on the location of the user and the location of the service. More recent observations are available in [ZZL10], where the Zheng et al. also provide a more detailed analysis of failure probabilities. For instance, for a random set of 100 public Web Services accessed from 24 countries, the authors observe a mean unavailability of 0.045, with standard deviation 0.1732. The data set analysed in [ZZL10] is publicly available.

General random variables for the response-times of Web Services may be defined over a set of services, or over invocations to one specific service. The observation in [KR04] that the 96% quantile is at 2 s for the set of Web Services analysed in the study is an example for the former, while the finding in the same paper that response-times for the Amazon and Google Web Services vary between 502–510 ms and 329–1046 ms, respectively, is an instance of the latter. Another description of response-times as a random variable over invocations to the same service may be derived from [GKT+08], where histograms for two public Web Services are provided. We note that neither of these studies gives explicit fault models, and furthermore, that there is no description of the shapes of the probability distributions in [KR04]. Response-time measurements for Web Services are also available in [ZL08], where both mean and standard deviation are given, divided by Web Service and user location. Zheng et al. note that large mean values are typically correlated with large standard variation and high failure rates.

More elaborate models still describe response-times by random variables, but allow additional parameters to reflect operating conditions. Fault models of this type may be derived from the studies in [DGP05, JKB03, TVN+03, RWW09]: In [DGP05] and [JKB03] E-Commerce applications are studied in testbed environments. [DGP05] contains 90% quantiles for response-times at load levels of 50, 100, 150, and 200 clients simultaneously accessing the application. The results are split by the services that make up the application and by two different usage profiles. Depending on the load level and the usage profile (Browsing/Ordering), the Flights, Credit Check and Process Order services exhibit 90% quantiles of response-times of

150–200 ms/150–800 ms, 120–550 ms/150–700 ms, and 150–800 ms/200–900 ms, respectively. A characterisation of load-dependent response-times by averages is given in [JKB03]. For load varying between 10 and 100 users, mean response-times increase from 500 ms to 1800 ms (flattening out at 60 users due to saturation). The study of Web Services mean response-times in [TVN+03] shows that, starting at 140 requests per second, response-times rise steeply to slightly above 7000 ms at 160 requests/s, where a plateau is reached (due to saturation). Unfortunately, due to the scaling of the graph, response-times for less than 140 requests per second are not discernible. Finally, in [RWW09] we provide Acyclic Phase-type (ACPH) models for the response-times of the SUN Java Adventure Builder at load levels of 10 and 50 users. Furthermore, the effect of IP packet loss on response-times is studied. It is shown that load and packet loss both increase mean response-times and the variance of the response-times distribution.

We note that, with the exception of [RWW09], the studies give very sparse descriptions of response-time distributions. While one might assume that response-times follow an exponential distribution (in which case the mean would be sufficient for parameterisation), this seems doubtful from the findings in [RWW09]. To the best of our knowledge, the ACPH models in [RWW09] are the only explicit fault models for Web Service response-times, and even these only describe response-times by random variables and cannot capture variations over time. We thus observe that so far the behaviour of Web Services has not been captured sufficiently well by fault models. Therefore we now perform the next step in the recursion, by splitting up the Web Service into the platform on which it is deployed and the storage it requires for operation.

1.2.1.1 Platform

We use the term 'Platform' to refer to the entire physical system implementing the Web Service. That is, we now no longer view the Web Service as an abstract entity that responds to requests, but instead consider the actual computer system(s) involved. While we are not aware of studies that deal specifically with physical systems for Web Services, one may argue that these systems are similar to other complex computer systems. Based on this premise, we extend our view to include fault models for computer systems in general. However, we should keep in mind that these models are derived from the behaviour of physical systems that might differ from those underlying Web Services.

Bernoulli and general random-variable models of the end-user availability of large systems on the Internet can be parameterised based on [MP02]. Analysing service-availability for three web sites, the authors report availability values ranging from 0.9305 to 0.9994, depending on the assumed locations of faults. In particular, they state that local problems have a strong impact on availability. The presented graph

of response-time over availability may serve to parameterise a random variable representing timing-failures that amount to service failures.

Simple stochastic-process models for faults can be thought of as consisting of an 'ok' state and a 'failed' state. In the 'ok' state, the modelled system provides its intended service, while in the 'failed' state the service is not available. Parameters for this kind of model can be derived from [SG06], where failure data of systems in the Los Alamos National Laboratory (LANL) from the period 1996–2005 is analysed, from [OGP03], where three commercial Internet services are studied, and from [LMG95], where the reliability of Internet hosts is analysed.

In the model, sojourn times in the 'ok' and 'failed' states are characterised by the time-to-failure (TTF) and time-to-repair distributions (TTR), respectively. For the TTF analysis, the authors of [SG06] pick one system out of their data set. They treat early system life and remaining life independently, since this system has significantly higher failure-rates during early system life. Furthermore, as the chosen system in [SG06] is comprised of 49 physical nodes, they consider failures occurring in one single node and in the whole system. During early life, the TTF distribution for the single node has a squared coefficient of variation of $c^2 = 3.9$ and can be described by a lognormal distribution, while no standard distribution provides a good fit for data from the system-wide view. The latter finding is due to the high number of simultaneous failures, which imply failure interarrival-times of zero. During remaining system lifetime the TTF distribution for the single node has $c^2 = 1.9$ and is approximated well by Weibull (shape parameter 0.7) and Gamma distributions. Weibull (shape parameter 0.78) and Gamma distributions also describe TTF distributions in the system-wide view. The TTR distribution is obtained over all systems in the data set. This distribution has mean 355 min, median 54 min and $c^2 = 187$ and is approximated by a lognormal distribution. Mean and median repair times depend on hardware type. Furthermore, repair-time distributions also depend on the root-cause of the failure. Combined with the root-cause analysis also provided in the study, this may be used to further parameterise repair-time distributions. We note that, with the exception of the shape parameters for the Weibull distributions, the paper does not provide distribution parameters explicitly, but shows the CDFs of the data and the fitted distributions. Time-to-failure and time-to-repair distributions are often characterised simply by their means, i.e. the Mean Time To Failure (MTTF) and Mean Time To Repair (MTTR). Assuming exponential TTF and TTR distributions, MTTF and MTTR are sufficient to specify a stochastic-process model. MTTR values for Internet services can be found in [OGP03]. The study in [LMG95] provides distributions of MTTF and MTTR values over Internet hosts. These distributions are described both by statistical properties and plotted densities and distribution functions. According to their statistical properties, neither the empirical MTTF nor the empirical MTTR distribution are well-described by an exponential distribution.

We note that a stochastic-process fault model for a Web Services platform may be derived from [SG06], if one deems the systems studied therein sufficiently similar to the systems that Web Services are typically deployed on. The study in [OGP03], whose data sets come from systems supposedly more similar to typical Web Services, provides only limited insight, in that the TTF distribution is not studied at all, and the TTR distribution is only characterised by mean values, which seems insufficient, considering the finding in [SG06] that neither the TTF nor the TTR distributions are exponential. The analysis in [LMG95] provides some insight into the distribution of MTTF and MTTR values over hosts, but does not characterise Time To Failure (TTF) and Time To Repair (TTR) distributions for the hosts in more detail.

We now apply the next step of recursion, by dividing the platform of a Web Service into its components. We consider the application server and SOAP framework, and virtualisation software. One further component of interest would be the operating system.

Application Server/SOAP Framework The software implementing Web Services is typically deployed within an application server and a SOAP framework, which provide the run-time environment required by the service to run and communicate with other Web Services. We thus now consider the application server and SOAP framework as a server, and the Web Service as a client to this server. With respect to fault models the service is then the run-time and communication functionalities required by the service. Again, both unavailability (which would in turn render the Web Service unavailable to its clients) and timeliness are relevant properties.

A stochastic-process model for application-server failures may be derived from the Markov reward model for the availability of a dependable application server presented in [TKDT04]. To this end, we consider the application server unavailable (unable to provide the run-time environment for the Web Service) if the high-level model is in one of the two 'failure' states. Transitions into and out of the 'failure' states are then described by exponential distributions, whose rates have been computed based on sub-models which are parameterised partly based on measurements.

Models of SOAP framework throughput (served requests per second) as functions of time under heavy load might be obtained from the study of the SOAP framework Axis 1.3 in [SMS06]. There, it is shown that memory leaks in the framework result in decreasing throughput, and even application-server crashes. From the figures, it appears that, depending on the load, linear functions with a strong negative slope or exponential functions may approximate the throughput degradation well. Unfortunately, the authors have not attempted fitting analytic functions to their data. We note that, although it appears likely that the causes of these memory leaks have been addressed in later versions of Axis, one may still employ these observations as guidelines when studying the effects of memory leaks, and software aging in

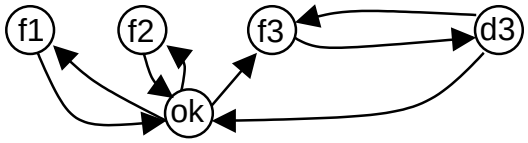


Figure 1.2: CTMC fault model (incomplete) for virtual machine reboots.

general. Note that the data given in [SMS06] represents only single observations at each time-point. Nonetheless, it may indicate how a stochastic process could describe the effects of memory leaks.

Virtualisation Recent years have seen a resurgence of virtualisation as a technology for increasing utilisation of expensive hardware and reducing power consumption. When virtualisation is used, many server instances may run on a single physical machine. With service-oriented systems, this implies that many services might be running on the same physical system, each within its own application-server environment. The required virtualisation software, however, adds another layer of complexity, and hence another source of faults, with respect to both timeliness and availability. While we are not aware of studies focussing on virtualisation software behaviour, it has been pointed out in [KC07] that faults, and fault-prevention by rejuvenation of the virtualisation software may require that virtual machines be stopped and restarted. The way in which these restarts affect the virtual machines depends on the virtualisation software, and may differ from normal reboots on physical hardware.

A stochastic-process model for the effects of virtual-machine reboots can be derived from [KC07]. The model (Figure 1.2) has one 'ok' state, in which the virtualisation software has no effect on the system running in the virtual machine, and three failure states ('f1' through 'f3'), in which the virtual machines are down. Additionally, there is one degraded state ('d3'), in which the machines are up, but provide service at reduced throughput. The three failure states reflect the different machine resume strategies considered in [KC07], which differ in their effects on the systems running in the virtual machines. Failure states 'f1' and 'f2' represent 'resume from disk' and 'resume from RAM', and these strategies differ only in the incurred downtimes. The 'f3' state is used for the 'cold reboot' strategy. This mechanism, besides incurring a delay, also results in a state where throughput of the systems inside the virtual machines is degraded, since caches are emptied during machine reboot. The authors of [KC07] provide measurements for the downtimes, and also for the amount of throughput degradation. Assuming that downtimes follow an exponential distribution, we may use these times as parameters in our model. Throughput degradation provides a guideline to describe behaviour in the degraded

state. Note that the model is incomplete, in that for some transitions (notably the ones *into* the failed states) no characteristics are given. This is due to the fact that [KC07] does not focus on the failure behaviour of virtualisation software and instead concerns the effect of the resume strategies.

Virtual machines may be migrated between hosts, e.g. to improve performance by exploiting locality properties, or to improve overall resource utilisation. The time required for the migration of virtual machines and the effects of migration on performance are studied by [ZF07]. Migration is split into resume, copy, and suspend phases. Performance is assessed as the impact on the time to complete an iteration of CPU-intensive and memory-intensive benchmarks. Furthermore, as an example for the effects on realistic workloads, the throughput of HTTP servers deployed inside the VMs is studied. Mean and standard deviation are given for various scenarios. These can be considered random-variable models of the virtualisation component.

1.2.1.2 Storage/Databases

Most services rely on extensive data manipulation and storage, e.g. for storing and retrieving service or business-process state. From the perspective of a service-oriented system, the service of databases, or storage in general, is to store and retrieve data.

Bernoulli models for the service availability of database systems can be derived using [CRM00] and [VM03]. [VM03] give availability estimates for a database system in normal operation and under the influence of operator faults. Depending on hardware and software platform, database version and database configuration options, availability observed at the server/client, respectively, varies between 74.2/68.7% and 93.5/83.9%, with database configuration having the largest impact. The probability of low-level hardware/software faults causing database failures, studied in [CRM00], may be used to parameterise a model for database faults, if the probability of low-level faults is known.

A random-variable model for the throughput of database systems without and with injected operator faults is the second type of model that can be derived from [VM03]. Unfortunately, throughput is only specified by mean values. Throughput depends on hardware and database configuration, varying between 1411 and 4394 transactions per minute without faults, and 896–3043 transactions per minute with faults.

Recovery-times of a database system with two types of automatic recovery are provided in [VM02]. The data is given as mean values for the recovery-time under operator faults and with different database configurations. These may be used as a parameter in a stochastic-process model, however, there is no characterisation of the distribution of the recovery-times, and no times-to-failure are given.

A stochastic-process model of the occurrence of undetected disk errors (UDEs) in RAID systems is available in [RBD+09]. Based on data obtained by [BGPS07]

and [SG07], [RBD+09] derive a discrete-event model of UDEs. The model allows to reproduce the occurrence of UDEs in RAID systems, dependent on the workload. It should be noted, however, that, due to its complexity, the model may be too expensive to be used in test-beds.

The papers only present a first glimpse into the behaviour of storage systems, as seen by users of the systems. While these data may be used to derive fault models, this area could greatly benefit from further analyses of such systems.

1.2.2 Communication

As service-oriented systems rely on communication between services, faults in the communication adversely affect service quality. We split this part into faults in the Web Services stack and in the Internet, where the former subsumes all technologies specific to Web Services, and the latter has a broader focus on the faults in the communication technologies that enable Web Services communication, but are not specific to them.

1.2.2.1 Web Services Stack (Infrastructure)

The first category of interest is the Web Services Infrastructure, i.e. those components of a service-oriented system that enable service communication. This category contains service-repositories, infrastructure for service orchestration (e.g. BPEL engines), SOAP proxies, etc. Unfortunately, we are not aware of studies specifically focussing on Web Services infrastructure components. These components may be seen as services, and consequently the fault models presented in the previous sections may be used. However, more research effort should be directed towards understanding the behaviour of the Web Services infrastructure, and on providing fault models for it, as failures of the infrastructure are likely to have a heavy impact on quality of service of the system.

1.2.2.2 Web Services Stack (Transport)

Next, our focus lies on the Web Services stack, i.e. on the communication protocols specific to Web Services. In Figure 1.3 we show an instance of the SOAP stack; we now consider all layers above the lowest two (HTTP and TCP/IP), which are not specific to Web Services or service-oriented systems.

The Web Services Reliable Messaging Protocol (WSRM, [BIMT05]) is an addition to the SOAP stack that provides reliable SOAP message transmissions over unreliable SOAP transports. A set of Acyclic Phase-Type Distribution (ACPH) models for the message-transmission times for several WSRM timeout strategies and several levels of IP packet loss are described in [RW08b]. The measurement data shows that, even though both TCP and WSRM mask the effects of IP packet loss, there

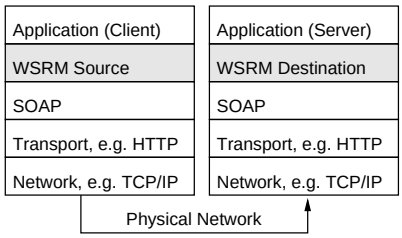


Figure 1.3: SOAP Stack

are still characteristic patterns of the transmission-time distribution (viz. spikes at the values of the TCP RTO timeout) in the message transmission times, which are also captured in the models.

We note that, so far, only very few models for the behaviour of communication components of service-oriented systems are available. Since Web Services typically communicate over the Internet, or networks using Internet technology, their communication is affected by the behaviour of these networks. This has long been a field of intense study, and thus we may hope to find appropriate models that could be used in the study of SOA systems. As with the Web Services stack, we divide this part into infrastructure, which refers to those components enabling the communication, and transport, which is the actual transmission of upper-layer messages or data packets.

1.2.2.3 Internet (Infrastructure)

Here, we consider two important infrastructure elements of the Internet, viz. the Domain Name System (DNS) and the routing infrastructure. As both communication with the Web Services infrastructure and between services relies on DNS lookups of the systems the services are deployed on, faults in the Domain Name System may affect both availability and timeliness of Web Services communication: If name lookup fails, then a Web Service may be invisible (and thus unavailable), and delays in lookups lead to delays in the communication.

Bernoulli models for DNS server availability can be parameterised using [PHA+04], where detailed availability values are given for name servers. In particular, the mean availability over all analysed servers is 0.9785, with median 1, i.e. for most servers no unavailability period could be observed, while those with outages tend to have rather low availability.

Random-variable descriptions of DNS lookup response-times from correctly configured servers, and from servers with one of three different misconfigurations can be derived from [PXL+04]. The paper provides CDFs of these random variables. In particular, non-responding servers increase mean response-times from about 60 ms

to 3 s, and up to 30 s in extreme cases. The paper also analyses the frequency of misconfigurations at various top-level domains, and thus gives the probability of hitting a misconfigured server, if domains are chosen at random. Unfortunately, no information on temporal dependencies, or variations in responses for individual addresses is provided.

A stochastic-process model for service availability of DNS servers is the second contribution in [PHA+04]. The model consists of an 'ok' and a 'failure' state, with the former handling DNS requests as expected and the latter not responding to requests. The transitions between both states are described by the time-to-failure and time-to-repair distributions, which are characterised by mean, standard deviation and median, and by graphical CDFs. The parameters differ depending on the analysis method; however, it appears that time-to-failure (TTF) distributions are hypo-exponential, while the recovery-time/time-to-repair (TTR) distributions are hyper-exponential. For instance, one TTF distribution has mean 125.9 h and standard deviation 99.1 h, while the associated TTR distribution's mean and standard deviation are 7.2 h and 9.5 h, respectively. This means that, in relation to the mean, TTF distributions exhibit low variance, whereas the variance for TTR distributions is relatively large. However, note that time-scales of TTF and TTR times differ widely, as evidenced by the means.

The routing subsystem is fundamental to all communication in the Internet. With respect to Web Services, we expect routing failures to manifest as inability to communicate with a service, i.e. service unavailability.

Based on the study of Internet routes in [LAJ99] both Bernoulli and stochastic-process models of the routing system can be parameterised. The paper provides CDFs for the availability of routes, and CDFs for the MTTF and MTTR of route failures. All CDFs are over routes, i.e. specify the proportion of routes that exhibit a particular property. It is stated that the MTTF of the majority of routes is 25 days, with MTTR of at most 20 min.

1.2.2.4 Internet (Transport)

We now consider transport in the Internet. As an abstract concept, the Internet itself is a single server whose service is data transfer, and where deviations could be both unavailability and delays of data transfer.

A two-state stochastic-process model for the availability of the Internet is presented in [DCGN03a]. In the 'available' state, two hosts on the Internet are able to communicate, while in the 'unavailable' state the hosts cannot communicate. The model is parameterised based on several measurements in the period 1994–2000. The time-to-failure distribution is assumed to be exponential, with a mean of

48111 s. The time-to-repair distribution for unavailability events longer than 30 s[2] is approximated by the Pareto distribution with CDF $R(t) = 1 - 19t^{-0.85}$. Truncated at 500,000 s, this distribution has a mean of 609 s. We note that the study in [MP02] (see Section 1.2.1.1) may also be employed for deriving Bernoulli and general random-variable models for Internet availability. However, in contrast to [DCGN03a], in [MP02] the focus lies on host availability, not on network availability.

Let us now look at the protocols that the SOAP stack depends on. As the dominant SOAP transport in today's service-oriented systems, HTTP will be of interest. We then descend further in the SOAP stack (Figure 1.3) and investigate the TCP/IP level, which underlies the HTTP transport.

HTTP When SOAP over HTTP is used without any further protocols (such as WSRM), SOAP may behave almost identical to HTTP, as far as its reliability and timeliness properties are concerned. There will be overhead due to SOAP processing, but SOAP does not add any reliability mechanisms.

Based on [NA04, RvMW04, RvMW06b], general random variables can be used to describe response-times of HTTP connections. [NA04] provide parameters for Pareto distributions that fit file-transmission times in the Web, noting that file sizes and network conditions only affect the location parameter, while the shape parameter is only affected by server load. The analyses in [RvMW04, RvMW06b] show that IP packet loss may have a strong impact on HTTP transmission times, furthermore, the response-times for subsequent attempts to the same location are often not correlated.

To the best of our knowledge, no models that capture temporal behaviour of HTTP response times exist.

TCP/IP Findings in the papers from the previous section illustrate that low-level failures affect quality of service, even though they are commonly masked by higher-level protocols, especially TCP, which provides reliable transport over links that exhibit IP packet loss. TCP's fault-handling mechanism for packet loss results in transmission delays, which then affect QoS. This section thus focusses on models for IP packet loss. Note that, while models for packet-loss may be difficult to introduce into abstract models of service-oriented systems, they can be easily employed in test-beds, where they allow the study of the complex timing-behaviours resulting from the interaction of the various fault-handling mechanisms in the SOAP and TCP/IP stack. From these, abstract models for higher-level behaviour can be obtained (cf. [RW08b, RWW09]).

[2]Unavailability events shorter than 30 s are excluded, since they are considered irrelevant for the techniques studied in the paper ([DCGN03b], p. 9).

Bernoulli models for packet loss can be parameterised based on [ZDPS01, PAB+05, CWA+05]. According to [ZDPS01], 40% of the low-loss traces, but only 6% of the high-loss traces analysed can be described by Bernoulli packet loss. This may indicate that for high loss levels Bernoulli models are not appropriate. Only 1% of the traces analysed in [ZDPS01] have loss rates larger than 10%. [PAB+05] remark that even in a LAN environment packet-loss rates may exceed 1%.

Stochastic-process models for packet loss available in the literature may be divided into Markovian, i.e. variants of the Gilbert-Elliott model, and non-Markovian models. Both describe temporal dependencies in packet loss by at least two states with differing loss probability, and characterisations of the transitions between the states. Furthermore, one may distinguish between models that describe single-loss events (i.e. loss of one packet) and loss-episodes (i.e. consecutive loss of packets, or periods of elevated loss). The study of constancy of Internet properties in [ZDPS01] concludes that packet-loss in change-free regions is well approximated by Markovian models: With low loss, 89% of traces are described by the two-state Gilbert model, 98% by a three-state Gilbert model, and 99% by four-state Gilbert models. For lossy traces, the numbers are 6%, 68%, 85%, and 96%. Furthermore, loss occurs in bursts of consecutive losses, not in losses of nearby packets. This argues for models with loss probabilities of 0 and 1 in the loss-free and lossy states, respectively, and consequently a description of the loss process as interchanging loss-free periods and loss episodes. Loss-episode interarrivals are independent and identically distributed (IID) and well-modelled by exponential distributions at timescales up to 10 s. Beyond timescales of 10 s the authors observe correlation, which they attribute to a mixture of loss-processes at these timescales. The paper contains some plots of CDFs of inter-loss-episode- and loss-episode-lengths. Similar findings are presented in [YMKT99]. The analysis in [HH08] provides explicit parameters for three variations of the Gilbert model: Simple Gilbert (loss probability 0 and 1), Gilbert (loss probabilities 0 and $1-h$) and Gilbert-Elliot (loss probabilities $1-k$ and $1-h$). The models are fitted to data on multiple timescales. The data was obtained by feeding a trace into a bottleneck and observing packet loss due to tail drop.

Non-Markovian models for packet loss can be characterised based on [HNA02, BS03, BS04, HS08]. In [HNA02] interloss-times for non-congestion-related loss are described by a mixture of exponential distributions for intra-loss-burst times (with a length below 1 s), and a Normal distribution for the inter-loss-burst times (with mean 10 s). The authors propose a six-state Markov-Modulated Poisson Process (MMPP) to describe the inter-loss times. One state of the model is used to reproduce intra-loss behaviour. Unfortunately, the authors omit the parameters for their model. The studies in [BS03, BS04] contain plots of CCDFs for the lengths of loss-free and lossy periods (regrettably omitting more detailed characterisations of the loss process, as the paper's focus is on a comparison of probe- and SNMP-based measurements). A

	Bernoulli	Random Variable	Stochastic Process
Service	[KR04, SF07, ZL08, ZZL10]	[KR04, GKT+08, DGP05, JKB03, TVN+03, RWW09, ZL08]	—
Platform	[MP02]	—	[SG06, OGP03, LMG95, MP02]
	—	—	—
Application Server/ SOAP Framework	—	—	[TKDT04, SMS06]
Virtualisation	—	[ZF07]	[KC07]
Storage/Databases	[CRM00, VM03]	[VM03]	[VM02, RBD+09]
Communication	—	—	—
Web Services Stack	—	—	—
WS Infrastructure	—	—	—
WS Transport	—	[RW08b]	—
Internet Infrastructure	[LAJ99, PHA+04]	[PXL+04]	[LAJ99, PHA+04]
Internet Transport	[ZDPS01, PAB+05, CWA+05, MP02]	—	[DCGN03a, NA04, RvMW04, ZDPS01, YMKT99, HH08, HNA02, BS03, BS04, HS08]

Table 1.1: Classification of fault models for service-oriented systems by architectural level and fault model type.

Weibull distribution with shape parameter 2.9 and scale 0.3 is used to describe the loss-interarrival times in [HS08].

1.2.3 Major Observations

Table 1.1 provides an overview of the papers containing fault models reviewed in our study, classified by the components of a service-oriented system and the type of fault model. We observe that the coverage by fault models differs between the levels. For instance, while the Internet Transport level appears well-represented, there are no models for an abstract Communication between Services available. The degree of coverage also differs with respect to the types of models that are available. This is particularly regrettable for the Service level, whose only descriptions are general

random variables (which are often not even specified very rigorously, cf. e.g. Section 1.2.1). Here, stochastic-process models that also describe changes in faults over time would be of great use, since fault models of Web Services may be useful in answering many research questions related to service-oriented systems.

1.3 SOA Fault Classification Schemes

In assessing QoS aspects of service-oriented systems, fault-classification schemes serve three important purposes. First, they help to structure the study of available fault models, second, they provide a structure for applying the fault models in the system model, and thus, third, a structured way for developing models for SOA systems that are not limited to modelling a specific system, but instead represent general properties.

We already applied a fault-classification scheme to structure our survey in the previous section. This scheme was inspired by architectural properties of SOA systems. Several other classification schemes for faults in service-oriented systems can be found in the literature. In this section we provide an overview of these. For each scheme, we first summarise its important aspects. We then illustrate and discuss its use in classification of fault models by applying it to the models surveyed in the first part of the chapter. The classifications thus derived serve the second purpose identified above.

1.3.1 Phase/Level Classification

We begin with the classification scheme in [CFR03], where faults are distinguished along a time dimension and an architectural dimension. In the time dimension, service-oriented systems are considered *phased-mission systems*, i.e. systems whose operational life consists of multiple phases with differing characteristics and requirements. [CFR03] argue that this view is correct when considering the experience of a single user, where non-overlapping phases may be distinguished. Globally, however, phases tend to overlap. The following phases are considered typical: *Infrastructure discovery* and *client registration*, where the user discovers and enters the physical environment; *service registration*, *service discovery/lookup* and *service selection*, in which first the provider publishes the service, followed by the client discovering and then selecting and invoking the service; and *system configuration* and *service delivery*, comprised of the infrastructure and the service dynamically adapting to meet the client's requirements, and finally the delivery of the service to the client. In the architectural dimension, three levels are identified: The *end-point level*, which contains client- and server-side application objects; the *service delivery level*, comprising the entities concerned with delivering the service; and the *lookup/discovery*

level, consisting of the service registry and the discovery infrastructure. Additionally, the *network level* is mentioned, but not described in detail.

We may attempt to use the classification scheme of [CFR03] as a means to order the papers discussed in the survey comprising the first part of this chapter. This then allows us to easily look up relevant papers for a given time-phase and architectural level. In order to do so, the following assumptions on our part are necessary: We interpret the End-point Level as equivalent to our high-level Web Service concept, the Delivery Level as the Platform and Storage levels, the Lookup/Discovery Level as the Web Services Infrastructure, and the Network level as a combination of our WS Stack and Internet categories.

In the Infrastructure-Discovery phase, faults in the DNS have a strong impact. These, analysed in [PXL+04], belong to the Network Level. For the Service-Discovery/Lookup phase, the analysis of UDDI entry validity in [KR04] provides a glimpse into faults at the Lookup/Discovery Level, while DNS faults ([PXL+04, PHA+04]) again affect the Network Level. Turning our attention to the Service-Delivery phase, we note that the distinction between the End-Point and Service-Delivery levels becomes fuzzy, since the faults discussed in [KR04, DGP05, JKB03, TVN+03, RWW09, TKDT04, ZL08, SF07, ZZL10] cannot be clearly attributed to either of the two. Assuming that the Service-Delivery Level is equivalent to our Platform and Storage levels, however, we can at least assign [SMS06, ZF07, KC07, OGP03, SG06, LMG95, CRM00, VM03, VM02, RBD+09] to the Service-Delivery Level. At the Lookup/Discovery Level, [KR04] again provides insight into UDDI faults. Finally, the studies in [RW08b, PXL+04, DCGN03a, DCGN03b, NA04, RvMW04, RvMW06b, ZDPS01, PAB+05, CWA+05, HNA02, BS03, BS04, HS08, HH08, PHA+04], which are concerned with faults affecting communication, belong to the Network Level at the Service-Delivery phase. The resulting classification of the surveyed papers is shown in Table 1.2.

As pointed out in [CFR03], the phase-level scheme has the advantage of capturing the fact that faults may have quite different effects, depending on when and where they occur. Note, however, that in Table 1.2 many rows are empty. This implies that at the respective phases none of the surveyed papers describe faults in any of the architectural levels. On the other hand, some categories overlap, in that faults cannot be clearly assigned to either category. In fact, one might argue that many of the faults considered in the surveyed papers affect most or all of the time-phases and levels, since in each of the phases and at all levels the involved systems are often similar to each other. Database faults, for instance, may have an impact in every phase and at every level where storage and lookup of data is required; and IP packet loss affects all of the time-phases in service-oriented systems, since these systems rely on communication between their distributed components. While the overlap between categories may be partly due to a lack of rigour in the definition

	End-Point Level	Service-Delivery Level	Lookup/Discovery Level	Network Level
Infrastructure Discovery	—	—	—	—
Client Registration	—	—	—	—
Service Registration	—	—	—	—
Service Discovery/Lookup	—	—	[KR04]	[PXL+04, PHA+04]
Service Selection	—	—	—	—
System Configuration	—	—	—	—
Service Delivery	[KR04, DGP05, JKB03, TVN+03, RWW09, TKDT04, ZL08, SF07, ZZL10]	[KR04, DGP05, JKB03, TVN+03, RWW09, TKDT04, SMS06, ZF07, KC07, OGP03, SG06, LMG95, CRM00, VM03, VM02, RBD+09, MP02, ZL08, SF07, ZZL10]	[KR04]	[RW08b, PXL+04, DCGN03a, DCGN03b, NA04, RvMW04, RvMW06b, ZDPS01, PAB+05, CWA+05, HNA02, BS03, BS04, HS08, HH08, PHA+04]

Table 1.2: Phase/Level Classification Scheme [CFR03] applied to the studies in our survey.

of the time-phases and architectural levels, it also appears likely that the scheme is too fine-grained to really describe real-world faults.

1.3.2 Three-level Classification

[GMKR07] propose a scheme that classifies faults solely along the architectural dimension. In contrast to [CFR03], no time-dimension is considered. Three levels

Service Side	[KR04, DGP05, JKB03, TVN+03, RWW09, ZL08, SF07, ZZL10]
Client Side Binding	—
Network and System	[TKDT04, SMS06, KC07, SG06, OGP03, LMG95, CRM00, VM03, VM02, RBD+09, MP02, RW08b, PXL+04, DCGN03a, NA04, RvMW04, RvMW06b, ZDPS01, PAB+05, CWA+05, HH08, BS03, BS04, HNA02, HS08, PHA+04]

Table 1.3: Application of the Three-level Classification Scheme from [GMKR07].

are distinguished: *Client-Side Binding*, i.e. faults occuring in the client during the binding (and perhaps also invocation) stage; *Network and System*, which comprises faults affecting the communication between the services, including faults in the Internet infrastructure and in the hosting subsystem (e.g. the application server); and *Service*, describing faults that occur in the service itself during processing.

Again, we employ this scheme to structure the papers in our survey, in order to be able to quickly identify studies concerned with faults of a given category. We map the levels in [CFR03] to our categories as follows: The Service level is equivalent to our Web Service level, while the Network and System level comprises the remainder of the levels we distinguished. The Client-Side Binding level, however, does not correspond to any of our categories. This is due to the fact that it only describes programming faults, which have not been included in our survey. In fact, it appears that these faults might be better described as affecting the functional behaviour of the service. We obtain the classification shown in Table 1.3: [KR04, DGP05, JKB03, TVN+03, RWW09, ZL08, SF07, ZZL10] all belong to the Service-Side level, while the remainder are Network and System faults. Note that most papers belong to the third category. This is probably a drawback of the simplicity of the classification scheme, which only provides a very coarse granularity.

1.3.3 Time-Phase Classification

We now consider the scheme proposed by [BWM07a, BWM07b]. While the scheme in [GMKR07] only considers the architectural dimension, the taxonomy proposed in [BWM07a, BWM07b] may be seen as focussing solely on the time-phases dimension of [CFR03]. The taxonomy in [BWM07a, BWM07b] is similar to that in [CFR03], in that the authors identify five time-phases a service-oriented system has to traverse in order for the service to be provided. Faults are classified according to the phases, as follows: The *Publishing* phase comprises faults originating during service-registration by the service provider. Faults occurring when the client discovers the service affect the *Discovery* phase. The *Composition* and *Binding* phases describe faults during

service composition and binding to the required service; and the *Execution* phase contains all faults during execution of the service request. Each of the phases can be broken down further, and faults may be connected by the 'is-a' and 'causes a' relationships, thus allowing further refinement of the scheme.

In attempting to classify the fault models surveyed in Section 1.2 to the five time-phases described in [BWM07a, BWM07b] we first note that, due to the absence of an architectural dimension in the scheme, we cannot provide a mapping between the categories we used in our classification scheme and the time-phases of [BWM07a, BWM07b]. Second, we observe that most of the faults considered in the refinements of the time-phases only affect the functional behaviour of the service. In fact, with the exception of the 'Timed Out' fault, *all* faults in the Publishing, Discovery, Composition, and Binding phases relate to functional behaviour. In the fifth phase (Execution), both the 'Timed Out' and 'Service Crashed' faults describe non-functional behaviour, while the 'Incorrect Result' fault again relates to functional behaviour. Furthermore, all faults defined for the Publishing phase relate to functional behaviour. With this in mind, we cannot assign any of the papers we surveyed to the first phase, while the remainder of the phases can be affected by any of the faults studied in the papers we considered, since each of these faults may result in 'Timed Out' faults. Therefore, we do not provide a table for the time-phase classification scheme. Furthermore, we note that it appears likely that the concept of architectural levels is necessary in order to be able to meaningfully classify fault models.

1.3.4 Multi-dimensional Classification

A multi-dimensional classification scheme is presented in [CBS+07]. The scheme is an application of the general concepts described in [ALRL04] to the field of service-oriented systems. [CBS+07] distinguish three broad categories of faults, and, orthogonal to these, three dimensions. The broad fault-categories are *Physical*, *Development* and *Interaction* faults, while the dimensions describe the phase of occurrence (during development/maintenance or during operation), the location where the fault occurs (internal or external to the system), and the architectural level (hardware or software).

Unfortunately, the authors recur to examples where rigid definitions of the categories might have resulted in more clarity. Nonetheless, based on the given examples, we may attempt the following mapping between the categories in our classification scheme and those in [CBS+07]: The Physical category comprises all levels below the Service and WS Stack. Development faults are faults on the Service level that affect functional behaviour, while non-functional behaviour is contained in the Interaction category. Then, we interpret the dimensions as follows: In the Phase dimension, all faults not directly and exclusively attributable to development or maintenance

	Physical	Development	Interaction
Development	[VM03]	—	—
Operation	[TKDT04, SMS06, KC07, MP02, SG06, OGP03, LMG95, CRM00, VM03, VM02, RBD+09, PHA+04, PXL+04, LAJ99, DCGN03a, NA04, RvMW04, RvMW06b, ZDPS01, PAB+05, CWA+05, HH08, HNA02, BS03, BS04, HS08]	—	[KR04, GKT+08, DGP05, JKB03, TVN+03, RWW09, RW08b, ZL08, SF07, ZZL10]
Internal	—	—	[KR04, GKT+08, DGP05, JKB03, TVN+03, RWW09, RW08b, ZL08, SF07, ZZL10]
External	[TKDT04, SMS06, KC07, MP02, SG06, OGP03, LMG95, CRM00, VM03, VM02, RBD+09, PHA+04, PXL+04, LAJ99, DCGN03a, NA04, RvMW04, RvMW06b, ZDPS01, PAB+05, CWA+05, HH08, HNA02, BS03, BS04, HS08]	—	—
Hardware	[SG06, CRM00, DCGN03a, NA04, RvMW04, RvMW06b, ZDPS01, PAB+05, CWA+05, HH08, HNA02, BS03, BS04, HS08, RBD+09]	—	[RW08b]
Software	[TKDT04, SMS06, KC07, MP02, SG06, OGP03, LMG95, CRM00, VM03, VM02, PHA+04, PXL+04, LAJ99, DCGN03a, NA04, RvMW04, RvMW06b, ZDPS01, PAB+05, CWA+05, HH08, HNA02, BS03, BS04, HS08]	—	[RW08b]

Table 1.4: Multi-dimensional Classification based on [CBS+07]

actions belong to the operational phase. The Internal category of the Boundary dimension contains all faults in our Service and WS Stack levels, while all faults below these levels are considered external. The Hardware/Software dimension does not directly map to any of our levels. We apply it by assigning fault models that

Physical Faults	[SG06, VM02, RBD+09]
Software Faults	—
Resource Management Faults	—
Communication Faults	[RW08b, PXL+04, LAJ99, DCGN03a, NA04, RvMW04, RvMW06b, ZDPS01, PAB+05, CWA+05, HH08, HNA02, BS03, BS04, HS08]
Lifecycle Faults	[CRM00, VM03, VM02, RBD+09]

Table 1.5: An Application of the Fault-Model Ontology from [LXM07].

exclusively describe hardware faults to the former category, while all others belong to Software.

Table 1.4 displays the result of assigning fault models from the surveyed papers to the categories of the multi-dimensional classification scheme of [CBS+07]. Note that the scheme (at least in our interpretation) is too coarse-grained to distinguish between most of the faults, as evidenced by the large number of papers that fall into the Physical category. Furthermore, the Development category is empty, since the fault models we consider only affect non-functional behaviour. Finally, note that (with our interpretation) the Internal/External categories of the Boundary dimension is equivalent to the Physical/Interaction categories.

1.3.5 Fault-Model Ontology

The last approach we consider was proposed in [LXM07]. This approach differs from the previous ones in that it provides a methodology, rather than a classification scheme. The methodology allows one to derive detailed descriptions of faults, with the ultimate goal of implementing test cases for these faults in a fault-injection testbed. In this approach, one starts with a coarse classification of faults, which is then successively refined until the faults of interest can be described sufficiently well for implementation in a test-bed. Thus the approach is similar to our concept of a fault model, albeit with a focus on test-bed models.

The initial classification consists of five elements: *Physical faults*, *Software faults*, *Resource Management faults*, *Communication faults* and *Lifecycle faults*. Unfortunately, with the exception of Software faults (for which an example is given), the categories are not described in detail. Still, we will try to apply them to our survey. In doing so, we assume that the Physical faults category corresponds to faults that can be uniquely identified as originating in hardware, and Lifecycle faults refer to operator faults. Table 1.5 presents the resulting classification. Note that the Software and Resource Management categories are empty; furthermore, many of the reviewed papers are missing from the table, since we could not identify an appropriate category.

1.3.6 Major Observations on Fault Classification Schemes

Several fault-classification schemes for service-oriented systems have been proposed in the literature. The phase-level scheme in [CFR03] considers both a time- and an architecture-dimension, thereby accounting for the fact that the effects of faults may differ, depending which phase they affect. The three-level scheme in [GMKR07] focusses solely on the architecture dimension, while the taxonomy in [BWM07a, BWM07b] instead considers only the time-dimension. The multi-dimensional scheme in [CBS+07] also considers architectural and time dimensions, splitting the former by referring to the system boundary and the difference between hardware and software faults. Furthermore, the scheme in [CBS+07] adds a distinction between functional and non-functional behaviour. Finally, the ontology in [LXM07] provides a mixture of an architectural dimension and an abstract concept of faults. Note, however, that neither [CBS+07] nor [LXM07] give sufficiently strict definitions of the respective categories, and thus our discussion of these approaches is based on our interpretation.

A number of observations can be made from our application of these approaches to the classification of the fault models surveyed in Section 1.2. First, it appears that a purely time-phase-based taxonomy may not be sufficient to distinguish real-world faults. This implies that some notion of system architecture is required. Second, one should be careful to choose the right granularity for the classification of faults. When categories are too fine-grained, as in [CFR03], many categories might not relate to observable faults. On the other hand, if the categories are too coarse (as in [GMKR07] and [CBS+07]), distinction between faults becomes difficult, since many faults are lumped into the same category. Third, strict and correct definition of fault categories is important. This is highlighted by the approaches in [CBS+07] and [LXM07], which required a large amount of interpretation before they could be applied to the surveyed studies.

We conclude this section by remarking that for the purposes of fault model classification the recursive approach we applied in Section 1.2 appears the most promising, even though it does not consider the time-dimension. If a time-dimension is required, the scheme in [CFR03], or a combination of this scheme with our scheme, may be most applicable. In cases where a coarser and purely architectural taxonomy suffices, one may apply the one from [GMKR07]. The schemes proposed in [CBS+07] and [BWM07a, BWM07b], however, do not appear well-suited to the classification of real-world faults.

1.4 Application: Fault Models in the Study of Service-Oriented and Massively Virtualised Systems

We now illustrate the use of our survey in two application scenarios. First, we will show how the fault models considered in the first part of the survey can be used in studying the effectiveness of restart strategies in service-oriented systems. Second, we address the application of these fault models beyond the SOA scenario. To this end, we consider massively virtualised systems typical for the emerging field of Cloud Computing.

1.4.1 Example: Restart in Service-Oriented Systems

Consider again our running example from the Introduction. We want to develop an algorithm for computing the request timeout in a service-oriented system. In service-oriented systems, transient faults can be due to common influences such as IP packet loss and temporary server overload [RvMW04, RvMW06a, RW08b, RWW09]. Here, response-times may be reduced by restarting at an appropriate time. Restarting too soon, or too often, however, increases load and may thus exacerbate the problem, increasing response-times even further. Therefore, the client should base the restart timeout on its operating conditions.

In order to evaluate the quality of restart algorithms at different abstraction levels, we may start by measuring performance in a test-bed with fault-injection, as was done for a Web Services Reliable Messaging (WSRM, [BIMT05]) implementation in [RvMW06a, RW08a]. WSRM is a protocol that provides reliable SOAP message transmissions over possibly unreliable transports, by retransmitting messages that have not been acknowledged by the recipient. In [RvMW06a, RW08a] we studied the effects of IP packet loss on the message transmission times experienced by an application employing WSRM. In order to do this, we needed models for IP packet loss. In [RvMW06a] we applied a Bernoulli model to generate uniform IP packet loss, while in [RW08a] we used a Gilbert model to model packet-loss correlations. Gilbert models are Stochastic-Process fault models according to our classification in Section 1.1.1. We pointed out that observations differed with these models in [RW08a].

Another way of studying optimal timeout-selection is the analytical approach presented in [vMW06]. With this approach, one considers the response-times of the service-oriented system as seen by the client. Response-times are modelled as a random variable. Therefore, in this approach we need a Random-Variable model of response-times at the Service level, such as the phase-type models presented in [RW08b, RWW09]. Note that these models differ in where they may be applied: As was already pointed out in Section 1.1.1, the models from [RW08b] can only be

used to study restart on the connection to the service. The models in [RWW09], on the other hand, are for response-times of the SUN Java Adventure Builder with IP packet loss and at various load levels. These models can be used to study optimal timeout-strategies for restart of requests to a service.

The studies above focussed on IP faults. However, using our survey, one may easily study the application of restart to address the effects of other faults, such as in the DNS, in the Service, or in routing. In order to do so, one would first define the fault of interest, and then, using Table 1.1, identify a layer and a publication where this fault has been modelled in an appropriate way.

1.4.2 Example: Performance Evaluations in Cloud Computing

Currently, Cloud Computing is one of the major trends in distributed systems. Following the definition of Buyya et al. in [BYV+09], Computing Clouds are characterised by the extensive use of virtual machines, flexible assignment of resources to users, utility pricing (e.g. pay per used CPU hour), use of Web Services standards, and central ownership and administration of the underlying hardware and software resources. Furthermore, physical machines within a Cloud are typically interconnected by dedicated high-capacity networks, while the connection to the user as well as between clouds (cf. [BLS+09]) and between clouds and external services relies on the public Internet. [BLS+09] additionally point out that virtualisation may go as far as representing physical resources only as abstract metaphors.

With the growing importance of Cloud Computing, performance and dependability issues in these systems also come into focus (cf. [KRB+12]). Various research questions may be defined, depending on the particular application area.

One interesting use for the flexible infrastructure provided by Cloud services is scientific computing, for which [OIY+09] pose the research problem of tuning scientific applications for efficient execution in a Cloud Computing environment. In their work, Osterman et al. show that computing performance may not be optimal with Cloud services, while, depending on the workload, I/O performance may be even better than with physical hardware (due to extensive caching). Communication-intensive applications may suffer from high latency between virtual machines. Furthermore, they find that virtualisation may lead to reproducible reliability issues.

[OIY+09] use a measurement-approach in a real Cloud service. Alternatively, one might want to use experimentation in a test-bed. However, as pointed out by [WBPG09], the complexity and costs of a Cloud environment are prohibitively large. Instead, they implement a simulation framework for studying Cloud performance. Their focus is on the MapReduce programming paradigm, which is popular for distributing parallelisable tasks onto nodes of massively virtualised systems. Wang et al. instrument the NS-2 network simulator to model different characteristics of a MapReduce setup. In particular, their approach allows to study the effects of

task, node, and link failures on the performance of the system. While their current implementation only supports simple failures, the fault models gathered in our survey may be used to gain deeper insights into performance characteristics. For instance, we might be interested in the average performance of a setup completing several MapReduce tasks over an extended period of time. Then, we would look up models for the appropriate task, node, and link failures in Table 1.1, e.g. using the stochastic-process model from [DCGN03a] to include the temporal behaviour of link failures in the study.

Another simulation approach is presented by [BRC09]. The CloudSim Toolkit models, among other aspects, virtual machines and VM scheduling, storage, network, and computing resources. The CloudSim Toolkit currently does not support the injection of faults, however, apparently it would only require a few modifications in order to apply the fault models gathered in our study. The separation of a Cloud system into components employed by Buyya et al. is similar to our architectural classification scheme, and consequently it is easy to choose appropriate fault models for the different components. For example, faults due to the network may be introduced by e.g. using the response-time distributions from [RW08b, RWW09] to delay communication. Similarly, the findings in [KC07] may be applied to simulate delays caused by the virtualisation infrastructure.

An analytical approach for computing the response-time in a Cloud-Computing environment was proposed by [XP09]. This approach uses random variables with exponential distribution to model service-times distributions. Choosing random-variable models from the first row of Table 1.1 (e.g. the PH models we proposed in [RWW09]) instead, we might obtain more realistic results.

1.5 Concluding Remarks

In this chapter we discussed the application of stochastic fault models in system evaluation. We introduced the concepts of system models that are parameterised by fault models, we surveyed fault models for use in system models of service-oriented systems and presented an overview of fault-classification schemes for faults in SOA systems. In the first part we have observed that the description and understanding of faults in many components of typical SOA systems are lacking, both in the number of studies and in the detail of the available fault models, since many authors just analyse statistical properties of the data and do not provide explicit fault models. The second part showed that fault-classification schemes often show weaknesses in their actual application. Such shortcomings manifest as both overlapping and empty categories.

Nonetheless, we strongly believe that both parts of the survey will help to improve the understanding of faults in service-oriented systems and their effects on

quality of service. We anticipate three major applications. First, based on our survey of both fault models and fault classification schemes, researchers can build and evaluate system models that reflect general properties of SOA systems. By replacing the common approach of modelling specific systems for specific studies, such system models will foster analyses and experiments that yield general, comparable results for whole classes of systems. Second, we expect that the gaps we identified in our survey will direct research efforts towards a better understanding of important components of service-oriented systems, such as the SOA infrastructure. Third, as illustrated in Section 1.4.2 using Cloud Computing as an example, the fault models we surveyed will help to evaluate and improve quality-of-service attributes of a wide spectrum of emerging parallel and distributed computer systems.

Part II

Phase-type Distributions

Chapter Two

Background on Phase-type Distributions

In this chapter we give an introduction to phase-type (PH) distributions. We define the notation that will be used throughout this work and summarise important results on their properties. The goal of this introduction is two-fold: First, we want to provide a concise summary of existing work on the theory behind PH distributions to the reader not familiar with the area. Our focus in this will be on concepts that are relevant in system evaluation, and especially on those that are required in the later chapters of this work. Second, we need to define the terms and notation we will use to refer to these concepts, which have often been described using different terms, or different notation, by different researchers.

The chapter is structured as follows: We first define phase-type distributions and summarise several important properties. We then describe sub-classes of PH distributions that are of particular relevance in system evaluation due to certain desirable characteristics. In Section 2.3 we discuss important representations of PH distributions and the transformations between these representations. Section 2.4 introduces canonical representations for PH distributions.

2.1 Notation, Formal Definition and Properties

We start with an intuitive definition of a PH distribution: Given a Continuous-Time Markov Chain (CTMC) with one absorbing state, as shown in Figure 2.1, we may enter this chain at some state i with probability α_i. Now, as time progresses, state-changes will occur and different states will be visited, before the chain finally enters the absorbing state (state 7 in Figure 2.1). For each state that is visited, the time before going to the next state follows an exponential distribution. Thus, the time required to reach the absorbing state from an initial state i is a sum of samples

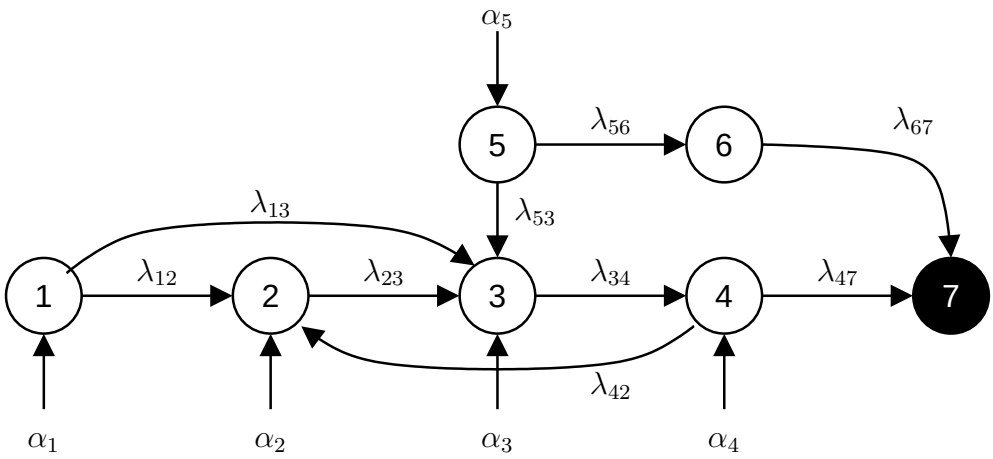


Figure 2.1: A continuous-time Markov chain with one absorbing state.

from exponential distributions. For a given CTMC, a phase-type distribution, is the distribution of the times to absorption that can be observed along the paths through the CTMC. This is summarised in the following definition:

Definition 2.1.1 (Phase-type distribution [Neu81])**.** *A continuous-time phase-type (PH) distribution is defined as the distribution of the time to absorption in a Continuous-Time Markov Chain (CTMC) with one absorbing state.*[1]

Definition 2.1.1 gives an intuitive definition of PH distributions. This definition serves to define the class, and will in fact be used repeatedly. However, we also need a formal description. In order to give this formal definition, we first need to introduce some notation: Let $\mathbb{1}$ be the column vector of ones, let $\mathbb{1}_i$ be the column vector whose ith entry is 1 and whose remaining entries are 0, and let $\mathbf{e}_i$ be the corresponding row vector. Furthermore, let $\mathbf{0}$ refer to the column vector, row vector, or matrix with all entries equal to zero. In order to avoid excessive notation, we do not specify the size of objects such as $\mathbb{1}$ or $\mathbf{0}$ explicitly, unless there may be ambiguities from the context. For the same reason, zero entries in matrices are usually left blank.

Following the definition as the distribution of absorption times in a CTMC, PH distributions are commonly represented by a vector-matrix tuple $(\boldsymbol{\alpha}, \mathbf{Q})$ that describes the transient part of the CTMC. The row vector $\boldsymbol{\alpha}$ gives the initial probabilities of the states in the transient part and the square matrix $\mathbf{Q}$ describes state-transitions prior to absorption. In this representation, $\boldsymbol{\alpha}$ is a non-negative row vector

[1] An equivalent definition using a Discrete-Time Markov Chain (DTMC) will yield the class of discrete-time phase-type distributions, which is not considered in this work.

whose entries sum to one, and $\mathbf{Q}$ is a sub-generator matrix for the transient states of a CTMC. We refer to a tuple with these properties as a *Markovian representation.*

Definition 2.1.2 (Markovian Representation)**.** *The vector-matrix tuple* $(\boldsymbol{\alpha}, \mathbf{Q})$ *is a* Markovian representation *of a PH distribution iff*

$$\begin{aligned} \boldsymbol{\alpha} &= (\alpha_1, \ldots, \alpha_n) \in \mathbb{R}^n && (2.1)\\ \boldsymbol{\alpha}\mathbb{1} &= 1 && (2.2)\\ \boldsymbol{\alpha} &\geq \mathbf{0}, && (2.3) \end{aligned}$$

and

$$\mathbf{Q} = \begin{pmatrix} -\lambda_{11} & \cdots & \lambda_{1n} \\ \vdots & \ddots & \vdots \\ \lambda_{n1} & \cdots & -\lambda_{nn} \end{pmatrix} \in \mathbb{R}^{n \times n} \quad (2.4)$$

is a non-singular matrix with

$$\begin{aligned} \lambda_{ii} &> 0 && (2.5)\\ \lambda_{ij} &\geq 0 \quad \textit{where } i \neq j && (2.6) \end{aligned}$$

for $i, j = 1, \ldots, n$ *and*

$$\begin{aligned} \mathbf{Q}\mathbb{1} &\leq_{el} \mathbf{0} && (2.7)\\ \mathbf{Q}\mathbb{1} &\neq \mathbf{0}., && (2.8) \end{aligned}$$

where $\leq_{el}$ *denotes the element-wise relation. With a Markovian representation,* $\boldsymbol{\alpha}$ *is the* initialisation vector *and* $\mathbf{Q}$ *is the* sub-generator matrix *of the associated CTMC with generator matrix*

$$\hat{\mathbf{Q}} = \begin{pmatrix} \mathbf{Q} & -\mathbf{Q}\mathbb{1} \\ \mathbf{0} & 0 \end{pmatrix}. \quad (2.9)$$

Definition 2.1.2 necessitates several remarks. First, note that we assume that with a sub-generator matrix of size n the absorbing state is the $(n+1)$th state. Other authors (e.g. [HT02]) assume that the absorbing state is at index 0. However, the definitions are completely equivalent.

Second, by allocating all probability mass to the transient states, our definition follows common practice in the area. This means that the absorbing state cannot be entered directly, i.e. the distribution does not have mass at zero. This convention does not pose a limitation, as mass can be assigned to zero by a simple rescaling of $\boldsymbol{\alpha}$ such that $\boldsymbol{\alpha}\mathbb{1} < 1$.

Third, as our use of the term 'representation' implies, there are usually several ways of representing a phase-type distribution. From Definition 2.1.1 it follows that all PH distributions have a Markovian representation, but this representation is usually neither unique, nor are all representations guaranteed to be Markovian. That is, a PH distribution may be defined by a tuple $(\boldsymbol{\alpha}', \mathbf{Q}')$ where $\boldsymbol{\alpha}'$ has negative entries or $\mathbf{Q}'$ is not a sub-generator matrix. Illustration of these statements must be deferred to Example 2.3.1 in Section 2.3, where we discuss the background for these examples.

Taking into account the last remark, we now give a definition of various characteristics of PH distributions that does not depend on the interpretation as a CTMC.

Definition 2.1.3 (Characteristics of phase-type distributions)**.** *Let* $(\boldsymbol{\alpha}, \mathbf{Q})$ *be a (not necessarily Markovian) representation of a phase-type distribution with* $\boldsymbol{\alpha} \in \mathbb{R}^n$*,* $\boldsymbol{\alpha}\mathbb{1} = 1$ *and* $\mathbf{Q} \in \mathbb{R}^{n\times n}$*. The* size *of the representation is the length of the vector* $\boldsymbol{\alpha} \in \mathbb{R}^n$*, which is equal to the size of the matrix* $\mathbf{Q}$*.*

The probability density function (PDF)*,* cumulative distribution function (CDF)*,* Laplace-Stieltjes Transform (LST) *of the CDF and* k*th* moment*, respectively, of the distribution given by* $(\boldsymbol{\alpha}, \mathbf{Q})$ *are defined as follows [Neu81, O'C90, HT02, TH02a]:*

$$f(t) = \boldsymbol{\alpha} e^{\mathbf{Q}t}(-\mathbf{Q}\mathbb{1}), \tag{2.10}$$

$$F(t) = 1 - \boldsymbol{\alpha} e^{\mathbf{Q}t}\mathbb{1}, \tag{2.11}$$

$$\tilde{F}(s) = \boldsymbol{\alpha}(s\mathbf{I} - \mathbf{Q})^{-1}(-\mathbf{Q}\mathbb{1}), \tag{2.12}$$

$$E\left[X^k\right] = k!\boldsymbol{\alpha}(-\mathbf{Q})^{-k}\mathbb{1}. \tag{2.13}$$

Phase-type distributions have been studied for a long time, and several important properties have been identified, often by different authors using different approaches. Many of these properties affect the application of PH distributions. In the following we list a few of these properties that pertain to the work presented here. The discussion is intentionally kept brief, as our goal is to provide the reader with a general understanding of these properties. We only give the statement and a short summary of the argument behind it, and refer the reader to the referenced literature for a much more complete treatment, including the formal proofs.

Theorem 2.1.1 (Positivity of PH distributions)**.** *The density* $f(t)$ *of a PH distribution is strictly positive for* $t > 0$*, i.e.*

$$f(t) > 0 \quad \textit{for} \quad t > 0. \tag{2.14}$$

The statement follows from the requirement that every PH distribution has a Markovian representation $(\boldsymbol{\alpha}, \mathbf{Q})$, as follows. First, note that the matrix exponential

in the density function (Equation (2.10)) can be written as a linear combination of exponential functions with positive coefficients (cf. [Ber09], Sec. 11.2, p. 711f for the details on the structure of the matrix exponential). For a Markovian representation $(\boldsymbol{\alpha}, \mathbf{Q})$ the coefficients of the elements of this linear combination in the density are the non-negative entries of the initialisation vector $\boldsymbol{\alpha}$ and the strictly positive entries of the column vector $-\mathbf{Q}\mathbb{1}$ (at least one element of $-\mathbf{Q}\mathbb{1}$ is greater than zero). Furthermore, since $\boldsymbol{\alpha}\mathbb{1} = 1$, at least one entry of $\boldsymbol{\alpha}$ is larger than zero. Finally, every exponential function in t is strictly positive for $t > 0$, and consequently $f(t)$ must be strictly positive for $t > 0$.

By relaxing the requirement that a Markovian representation exists and only requiring that (2.10) defines a proper density (i.e. $f(t) \geq 0$ for all t), one enters the class of Matrix-Exponential (ME) distributions. The same definitions as given in Definition 2.1.3 apply to this class, of which the PH distributions are only a sub-class.

Theorem 2.1.2. *Every PH distribution has rational Laplace-Stieltjes Transform (LST) of the form*

$$\tilde{F}(s) \quad = \quad \frac{P(s)}{Q(s)}. \tag{2.15}$$

If $Q(\rho) = 0$ for $\rho \in \mathbb{C}$ (i.e. ρ is a root of Q), we refer to ρ as a pole *of F. The poles of F are the eigenvalues of the sub-generator matrix $\mathbf{Q}$.*

The observation follows immediately from the expression given for the LST in (2.12). A more detailed discussion is available in [O'C90].

Theorem 2.1.3 (Closure Properties [Neu81])**.** *If, for $N \in \mathbb{N}^+$, $f_1(t), \ldots, f_N(t)$ are densities of PH distributions and $a_1, \ldots, a_N \in [0, 1]$ with $\sum_{i=1}^{N} a_i = 1$, then*

$$f_{Mixture}(t) \quad = \quad \sum_{i=1}^{N} a_i f_i(t) \tag{2.16}$$

$$f_{Sum}(t) \quad = \quad f_1(t) * f_2(t) * \cdots * f_N(t) \tag{2.17}$$

are also densities of PH distributions.

The above theorem states that the class of PH distributions is closed with respect to mixture and convolution of a finite number of distributions. Both can be easily shown by considering that, given the Markovian representations $(\boldsymbol{\alpha}_1, \mathbf{Q}_1), \ldots, (\boldsymbol{\alpha}_N, \mathbf{Q}_N)$, the respective Markovian representations for the mixture or convolution can be constructed by building the appropriate sub-generators [Neu81].

Theorem 2.1.4 (Lower bound on the squared coefficient of variation [AS87])**.** *The lower bound for the squared coefficient of variation (SCV) cv^2 of a PH distribution of size n is*

$$cv^2 \quad \leq \quad \frac{1}{n}, \tag{2.18}$$

where the equality holds for the Erlang distribution of length n.

In their aptly named paper [AS87], Aldous and Shepp show that the phase-type distribution with least variability is the Erlang distribution, which has $cv^2 = 1/n$. Any other PH distribution of the same size must then have an SCV of at least $1/n$. The lower bound on the SCV is the most general result for the moments of a PH distribution. Although there are several results for special cases (e.g. the moment bounds of the PH distributions of size $n = 2$ and $n = 3$ presented in [TH02b] and [HT07], respectively), similarly general bounds for higher moments are not known. In [HT08], Horváth and Telek provide numerical results for the moment bounds.

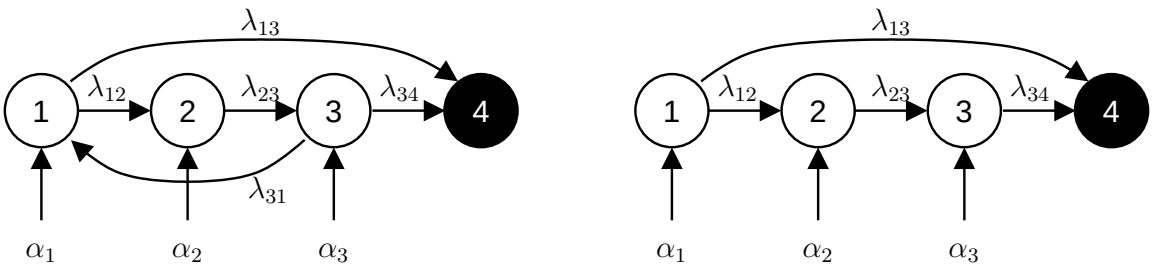
2.2 Classes of Phase-type Distributions

Several sub-classes of phase-type distributions have been defined over the years. The concept of PH *classes* is closely intertwined with the concept of PH *representations*. On the one hand, PH classes can often be conveniently defined as being the phase-type distributions of a certain structure. On the other hand, certain representations are only available for certain classes. We first discuss two important PH classes. Section 2.3 then introduces representations for these classes and sub-classes, and Section 2.4 discusses canonical representations.

Definition 2.1.1 defines the class of phase-type distributions. According to this definition, every CTMC with an absorbing state describes a PH distribution. A very useful subclass of PH distributions can be defined by only considering CTMCs without loops. This class is called the class of Acyclic Phase-type (APH) distributions, with the following formal definition:

Definition 2.2.1 (Acyclic Phase-type Distribution)**.** *A phase-type distribution is called* acyclic *if it has a Markovian representation $(\boldsymbol{\alpha}, \mathbf{Q})$ where $\mathbf{Q}$ is upper-triangular:*

$$\mathbf{Q} \quad = \quad \begin{pmatrix} -\lambda_{11} & \lambda_{12} & \dots & \lambda_{1n} \\ & -\lambda_{22} & \dots & \lambda_{2n} \\ & & \ddots & \vdots \\ & & & -\lambda_{nn} \end{pmatrix}. \tag{2.19}$$



(a) Phase-type distribution with cycles.

(b) Acyclic phase-type distribution.

Figure 2.2: CTMC representations for general and acyclic phase-type distributions.

Definition 2.2.1 is equivalent to requiring that an acyclic phase-type distribution has at least one underlying CTMC that is cycle-free. This restriction does not apply to the general PH class, where the underlying CTMCs of all representations may have any structure, as long as it contains an absorbing state. The difference is illustrated in Figure 2.2: The CTMC on the left contains a cycle, that is, a backward transition from state 3 to state 1. The CTMC on the right does not contain this transition and therefore there are no cycles and the CTMC describes an APH distribution.

One easily sees that the APH class is a true subclass of the PH class: The eigenvalues of an upper-triangular matrix $\mathbf{Q}$ as given in Definition 2.2.1 are the entries of the diagonal, $-\lambda_{11}, \ldots, -\lambda_{nn}$. Since $(\boldsymbol{\alpha}, \mathbf{Q})$ is Markovian, all entries of the matrix must be real, and consequently the sub-generator of an acyclic phase-type distribution cannot have complex eigenvalues. In Section 2.1 we remarked that the eigenvalues of the sub-generator give the poles of the Laplace-Stieltjes Transform (LST) of the distribution. Since the eigenvalues of the sub-generator of an APH distribution cannot be complex, the LST of this distribution cannot have complex poles. This restriction on the eigenvalues of the sub-generator does not exist for general PH distributions, whose sub-generator matrices may have complex eigenvalues, or, equivalently, whose LSTs may have complex poles. Consequently, the APH class is a true sub-class of the PH class.

A second important sub-class is the class of Hyper-Erlang distributions. As the name suggests, this class is defined as containing all distributions that are mixtures of Erlang distributions. Since Erlang distributions are PH distributions, mixtures of these distributions are again PH distributions. We give a formal definition of the Hyper-Erlang class based on representations in the next section. Here we only illustrate that, just as the APH class is a true sub-class of the PH class, the Hyper-Erlang class is a true sub-class of the APH class. In [TBT06], Thümmler et al. prove this statement using the squared coefficient of variation of the classes. Equivalently, we can apply the following argument: The Erlang distribution is certainly

a Hyper-Erlang distribution. Furthermore, it is also a member of the APH class, since it is a phase-type distribution without cycles. The Erlang distribution has identical rates in all transitions. On the other hand, the Hypo-Exponential distribution (or generalised Erlang) may have different rates on the transitions. Although a Hypo-Exponential distribution with differing rates is an APH, it is not an Erlang distribution, and thus also not a Hyper-Erlang distribution Consequently, the APH class contains at least one element that is not Hyper-Erlang, and thus is truly larger than the Hyper-Erlang class.

2.3 Transformations and PH Representations

A phase-type distribution is defined by a tuple $(\boldsymbol{\alpha}, \mathbf{Q})$. We remarked before that this representation is not unique, that is, one can find another tuple $(\boldsymbol{\alpha}', \mathbf{Q}')$ that defines the same distribution. $(\boldsymbol{\alpha}', \mathbf{Q}')$ may be of the same size or it may have a different size. We will now describe transformations between representations and illustrate the non-uniqueness of phase-type representations. With two representations that are of the same size, one may be computed from the other by a similarity transformation. The concept can be generalised to representations of different sizes.

We note that, while for every PH distribution there is a representation of larger size, representations with smaller size may only be found if the representation is non-minimal [Neu81] or if it is not PH-simple [O'C91]. Both concepts relate to the fact that a representation may contain redundant elements, e.g. states that are not visited or can be lumped (non-minimality) or unnecessary poles (non-PH-simplicity). In these cases the size of the representation may be reduced, by e.g. applying the methods from [Neu81, HZ06] or the pole-reducing method from [Pul09].

2.3.1 Transformations between representations

Similarity transformations between PH representations are based on the concept of similarity transformations between matrices, which are defined as follows:

Definition 2.3.1 (Similarity Transformation, Definition 3.4.4 in [Ber09], p. 188)**.** *Two matrices* $\mathbf{Q}, \mathbf{Q}'$ *are* similar *if there exists an invertible (i.e. non-singular) matrix* $\mathbf{S}$ *of the same size such that* $\mathbf{Q}' = \mathbf{S}^{-1}\mathbf{Q}\mathbf{S}$*. In this case,* $\mathbf{S}$ *is called a* similarity transformation matrix.

The following Lemma recalls two important properties of similarity transformations between matrices that will be used in the following. The proof is straightforward and uses the series expansion of $e^{\mathbf{Q}}$. It is given in Appendix A.

Lemma 2.3.1. *Let* $\mathbf{Q} \in \mathbb{R}^{n\times n}$ *and* $\mathbf{S}$ *be a similarity transformation matrix. Then*

$$(\mathbf{S}^{-1}\mathbf{Q}\mathbf{S})^k = \mathbf{S}^{-1}\mathbf{Q}^k\mathbf{S} \tag{2.20}$$

$$\mathrm{e}^{\mathbf{S}^{-1}\mathbf{Q}\mathbf{S}} = \mathbf{S}^{-1}\,\mathrm{e}^{\mathbf{Q}}\,\mathbf{S} \tag{2.21}$$

Similarity transformations between PH representations additionally require that the row-sums of $\mathbf{S}$ are equal to one: $\mathbf{S}\mathbb{1} = \mathbb{1}$. We can then compute the new initialisation vector $\boldsymbol{\alpha}'$ using the similarity transformation, as follows:

$$\boldsymbol{\alpha}' := \boldsymbol{\alpha}\mathbf{S}. \tag{2.22}$$

As summarised in the following lemma, $(\boldsymbol{\alpha}, \mathbf{Q})$ and $(\boldsymbol{\alpha}', \mathbf{Q}')$ are two representations for the same distribution :

Lemma 2.3.2 (Similarity transformation for PH distributions)**.** *Let* $(\boldsymbol{\alpha}, \mathbf{Q})$ *be a representation of a PH distribution with distribution function* $F(t) = 1 - \boldsymbol{\alpha}\,\mathrm{e}^{\mathbf{Q}t}\,\mathbb{1}$. *Given a non-singular matrix* $\mathbf{S}$ *with* $\mathbf{S}\mathbb{1} = \mathbb{1}$, *the tuple* $(\boldsymbol{\alpha}\mathbf{S}, \mathbf{S}^{-1}\mathbf{Q}\mathbf{S})$ *defines the same distribution.*

Similarity transformations are only applicable to representations of the same size. If the two representations are of a different size, we need a generalisation of the concept, which we obtain by using a non-square matrix $\hat{\mathbf{W}}$ instead of the square matrix $\mathbf{S}$ for the similarity transformation. The generalised similarity transformation is then described by the following lemma:

Lemma 2.3.3. *Let* $(\boldsymbol{\alpha}, \mathbf{Q})$ *be a representation of size* n *of a PH distribution with distribution function* $F(t) = 1 - \boldsymbol{\alpha}\,\mathrm{e}^{\mathbf{Q}t}\,\mathbb{1}$. *Let* $\mathbf{Q}' \in \mathbb{R}^{m\times m}$ *be a sub-generator matrix and let* $\hat{\mathbf{W}} \in \mathbb{R}^{n\times m}$ *such that* $\hat{\mathbf{W}}\mathbb{1} = \mathbb{1}$ *and* $\hat{\mathbf{W}}\mathbf{Q}' = \mathbf{Q}\hat{\mathbf{W}}$. *Then,* $(\boldsymbol{\alpha}\hat{\mathbf{W}}, \mathbf{Q}')$ *defines the same distribution as* $(\boldsymbol{\alpha}, \mathbf{Q})$.

Proofs for both lemmata can be found in Appendix A. Note that for $n = m$ similarity transformations and generalised similarity transformations are equivalent.

The following example illustrates the use of these transformations:

Example 2.3.1 (Transformations and Representations)**.** *Consider the PH distribution defined by* $(\boldsymbol{\alpha}, \mathbf{Q})$ *with*

$$\boldsymbol{\alpha} = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3}) \quad \textit{and} \quad \mathbf{Q} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & -3 \end{pmatrix} \tag{2.23}$$

and the similarity transformation matrix

$$\mathbf{S}_1 = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & \frac{1}{3} & \frac{2}{3} \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.24}$$

Application of $\mathbf{S}_1$ *to* $(\boldsymbol{\alpha}, \mathbf{Q})$ *yields the Markovian representation*

$$\boldsymbol{\alpha}\mathbf{S}_1 = (\frac{1}{9}, \frac{2}{9}, \frac{2}{3}) \quad \textit{and} \quad \mathbf{S}_1^{-1}\mathbf{Q}\mathbf{S}_1 = \begin{pmatrix} -1 & 1 & 0 \\ 0 & -2 & 2 \\ 0 & 0 & -3 \end{pmatrix}. \tag{2.25}$$

If we apply the similarity transformation matrix

$$\mathbf{S}_2 = \begin{pmatrix} 1 & 0 & 0 \\ 2 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.26}$$

instead, we obtain

$$\boldsymbol{\alpha}\mathbf{S}_2 = (1, -\frac{1}{3}, \frac{1}{3}) \quad \textit{and} \quad \mathbf{S}_2^{-1}\mathbf{Q}\mathbf{S}_2 = \begin{pmatrix} -1 & 0 & 0 \\ 2 & -2 & 0 \\ 0 & 0 & -3 \end{pmatrix}, \tag{2.27}$$

which is non-Markovian, due to the negative second entry of the vector $\boldsymbol{\alpha}\mathbf{S}_2$*, but still represents the same distribution. Application of the generalised similarity transformation matrix*

$$\hat{\mathbf{W}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.7 & 0.3 \end{pmatrix} \tag{2.28}$$

to $(\boldsymbol{\alpha}, \mathbf{Q})$ *results in the larger representation*

$$\boldsymbol{\alpha}' = \boldsymbol{\alpha}\hat{\mathbf{W}} = (\frac{1}{3}, \frac{1}{3}, \frac{7}{30}, \frac{1}{10}) \quad \textit{and} \quad \mathbf{Q}' = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 \\ 0 & 0 & -3 & 3 \\ 0 & 0 & 0 & -10 \end{pmatrix}. \tag{2.29}$$

Throughout this work we use these transformations to compute a new initialisation vector after we have changed the sub-generator matrix. Typically, we first construct a new sub-generator matrix and build a matrix $\hat{\mathbf{W}}$ that transforms the old matrix into the new matrix. The transformation matrix $\hat{\mathbf{W}}$ can always be obtained by solving the system of linear equations that defines it. In some cases where the

structure of the sub-generators is known, the required matrix can also be constructed in a straightforward way. After the transformation matrix has been built, we then compute the new initialisation vector using the above lemmata as $\boldsymbol{\alpha}' := \boldsymbol{\alpha}\hat{\mathbf{W}}$.

Similarity transformations are applied extensively in Chapter 4. A generalised similarity transformation provides the underlying argument for the proof of Theorem 2.4.2 sketched in Section 2.4.2.

2.3.2 Important Representations

Based on the structure of the Markov chain, different types of representations can be distinguished. These representations often help to reduce the number of free parameters in the representation and will form the basis of efficient random-variate generation and efficient PH fitting in later chapters of this work. In this section we describe several important structures. Our discussion follows the distinction into chain structures and branch structures, defined in the following:

Definition 2.3.2 (Chain and Branch Structures)**.** *Let* $(\boldsymbol{\alpha}, \mathbf{Q})$ *be a representation of size* n *of a phase-type distribution. The representation has* chain structure *if*

$$\mathbf{Q} = \begin{pmatrix} \mathbf{Q}_1 & -\mathbf{Q}_1\mathbb{1}\mathbf{e}_1 & & & \\ & \mathbf{Q}_2 & -\mathbf{Q}_2\mathbb{1}\mathbf{e}_1 & & \\ & & \ddots & \ddots & \\ & & & & -\mathbf{Q}_{m-1}\mathbb{1}\mathbf{e}_1 \\ & & & & \mathbf{Q}_m \end{pmatrix} \tag{2.30}$$

for matrices $\mathbf{Q}_j \in \mathbb{R}^{b_j \times b_j}, i = 1, \ldots, m$ *and* $\sum_{j=1}^{m} b_j = n$*. On the other hand, if*

$$\mathbf{Q} = \begin{pmatrix} \mathbf{Q}_1 & \mathbf{0} & & & \\ & \mathbf{Q}_2 & \mathbf{0} & & \\ & & \ddots & \ddots & \\ & & & & \mathbf{0} \\ & & & & \mathbf{Q}_m \end{pmatrix}, \tag{2.31}$$

then the representation is said to have branch structure*.*

The general structures just defined do not need to be Markovian. However, if $(\boldsymbol{\alpha}, \mathbf{Q})$ is a Markovian representation, they admit the following intuitive interpretation in terms of the CTMC: A representation in chain structure consists of subsequent blocks of states. Once the Markov chain has been entered at any of these blocks $\mathbf{Q}_j$, the absorbing state can only be reached by traversing all of the remaining blocks, $\mathbf{Q}_{j+1}, \ldots, \mathbf{Q}_m$. With a representation in branch structure, the blocks are in parallel

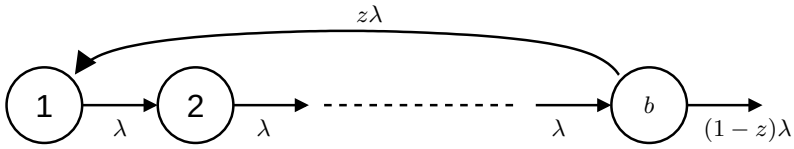


Figure 2.3: Structure of a Feedback-Erlang block with length b, rate λ, and feedback probability z.

and are not connected. Therefore, after entering block $\mathbf{Q}_k$, $k = 1, \ldots, m$, none of the other blocks can be visited.

Throughout this work we are particularly concerned with two special chain structures and two special branch structures where all block matrices $\mathbf{Q}_i$ have the same structure. For these representations, the blocks describe exponential, Feedback-Erlang, Erlang, or chains of Feedback-Erlang distributions. The Feedback-Erlang distribution is the most general block in these cases. This distribution was introduced by Mocanu and Commault in [MC99]. We first recall the definition of a Feedback-Erlang block and some of its important properties from [MC99], before discussing the special structures.

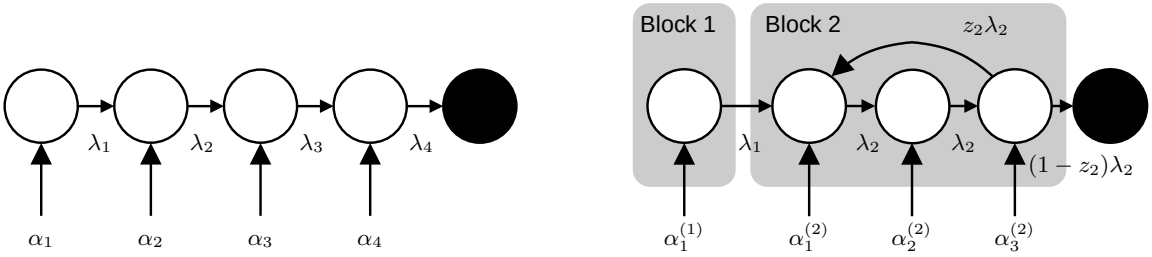
Definition 2.3.3 (Feedback-Erlang block [MC99])**.** *A* Feedback-Erlang (FE) block *is defined by the tuple* (b, λ, z)*, where* $b \geq 1$ *is the* length*,* $\lambda > 0$ *is the* rate*, and* $z \in [0, 1)$ *is the* feedback probability *of the block. The associated matrix* $\mathbf{F} \in \mathbb{R}^{b \times b}$ *has the following structure:*

$$\mathbf{F} = \begin{pmatrix} -\lambda & \lambda & & & \\ & -\lambda & \lambda & & \\ & & \ddots & \ddots & \\ & & & & \lambda \\ z\lambda & & & & -\lambda \end{pmatrix}. \tag{2.32}$$

Feedback-Erlang blocks with feedback probability $z = 0$ *or length* $b = 1$ *are often called* degenerate FE blocks*.*

Figure 2.3 illustrates the structure of a Feedback-Erlang block (b, λ, z) as defined by Definition 2.3.3 and the relation between FE blocks, Erlang and exponential distributions:. For $z = 0$, the Feedback-Erlang is simply the sub-generator matrix of an Erlang distribution of order b. An FE block with $b = 1$ is the sub-generator of an exponential distribution.

The importance of this structure lies in the fact that for $z > 0$ and $b > 2$ the block has at least one conjugate-complex pair of eigenvalues with non-zero imaginary part and that for every pair of conjugate complex eigenvalues an appropriate FE block can be constructed [MC99]. The following lemma summarises the results in [MC99]:



(a) Bi-diagonal representation.

(b) FE-diagonal representation.

Figure 2.4: Chain structures.

Lemma 2.3.4 (Eigenvalues of an FE block [MC99])**.** *The eigenvalues $\rho_1, \ldots, \rho_b$ of the Feedback-Erlang block defined by (b, λ, z) are*

$$\rho_k = -\left(1 - z^{\frac{1}{b}} \cos \frac{2(k-1)\pi}{b}\right) + \left(z^{\frac{1}{b}} \sin \frac{2(k-1)\pi}{b}\right) \mathrm{i}, k = 1, \ldots, b. \quad (2.33)$$

We now describe important chain and branch structures.

2.3.2.1 Chain Structures

We first define the bi-diagonal form and its natural extension, the FE-diagonal form. These chain structures play an important role as canonical representations and in the optimisation approaches we present in Chapter 4.

Definition 2.3.4 (Bi-diagonal representation)**.** *Let $(\boldsymbol{\alpha}, \mathbf{Q})$ be a representation in chain structure. The representation is* bi-diagonal *if $m = n$ and, for $i = 1, \ldots, m$,*

$$\mathbf{Q}_i \in \mathbb{R}^{1\times 1}, \text{ and} \quad (2.34)$$
$$\mathbf{Q}_i = \left(-\lambda_{ii}\right) \quad (2.35)$$

with $\lambda_{ii} > 0$. An alternative notation is $(\boldsymbol{\alpha}, \boldsymbol{\Lambda})$, where

$$\boldsymbol{\Lambda} = (\lambda_1, \ldots, \lambda_n) \in \mathbb{R}^n \quad (2.36)$$

is a row-vector of length n and $\lambda_i = \lambda_{ii}$ give the entries of the diagonal of $\mathbf{Q}$.

Figure 2.4(a) shows the CTMC for a Markovian bi-diagonal representation. Note that the CTMC can be entered at any state, but that the absorbing state can only be reached by passing through the remaining states. Familiar examples for distributions in bi-diagonal representation are the Erlang distribution, where $\boldsymbol{\alpha} = (1, 0, \ldots, 0)$ and $\lambda_i = \lambda_{i+1}$ for $i = 1, \ldots, n-1$ and the hypoexponential distribution, where the rates may be arbitrary.

The bi-diagonal representation is generalised by allowing the block matrices to be in Feedback-Erlang (FE) form. This generalisation results in the FE-diagonal form, defined below. Representations of this type have been first used in [MC99], where they are referred to as *Monocyclic* representations. The definition of a Monocyclic representation in [MC99] requires that the FE blocks are ordered according to their dominant eigenvalues (the dominant eigenvalue of an FE block is the largest eigenvalue of the corresponding matrix). We distinguish between the Monocyclic form and the FE-diagonal form, which does not require this ordering.

Definition 2.3.5 (FE-diagonal form)**.** *Let* $(\boldsymbol{\alpha}, \mathbf{Q})$ *be a representation in chain structure. The representation is in* FE-diagonal form *if all blocks* $\mathbf{Q}_1, \ldots, \mathbf{Q}_m$ *are in Feedback-Erlang form. An alternative notation for an FE-diagonal representation is* $(\boldsymbol{\alpha}, \boldsymbol{\Upsilon})$*, where*

$$\boldsymbol{\Upsilon} \quad = \quad \{(b_1, \lambda_1, z_1), \ldots, (b_m, \lambda_m, z_m)\} \tag{2.37}$$

describes the chain of FE blocks. Where appropriate, we apply the notation $\boldsymbol{\alpha}^{(j)}$ *to refer to the initial probability vector of the* j*th Feedback-Erlang block, i.e.*

$$\boldsymbol{\alpha} \quad = \quad (\boldsymbol{\alpha}^{(1)}, \ldots, \boldsymbol{\alpha}^{(m)}). \tag{2.38}$$

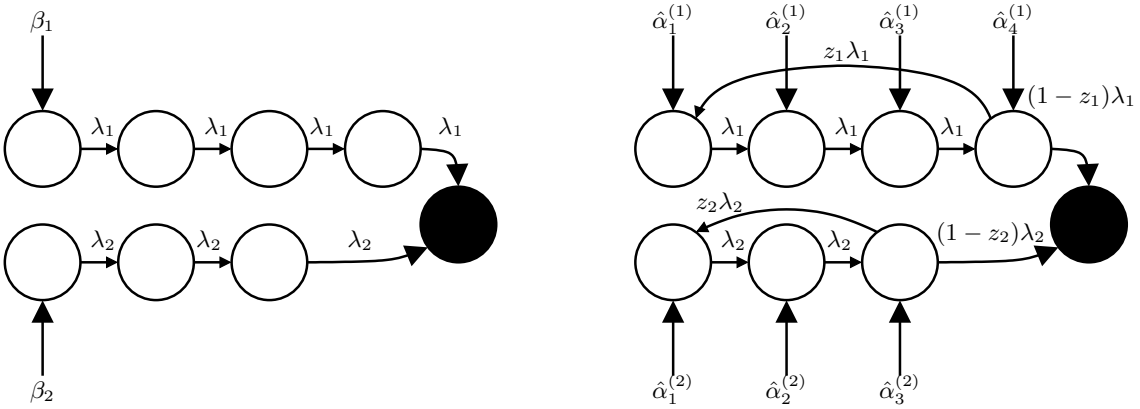
Formally, $\boldsymbol{\alpha}^{(j)}$ *is a row vector of size* b_j *with the following definition:*

$$\boldsymbol{\alpha}^{(j)} \quad = \quad (\alpha_{\sum_{l=1}^{j-1} b_l+1}, \alpha_{\sum_{l=1}^{j-1} b_l+2}, \ldots, \alpha_{\sum_{l=1}^{j-1} b_l+b_j}). \tag{2.39}$$

Figure 2.4(b) shows an example of the CTMC of a general PH distribution in FE-diagonal form. In this representation there are two Feedback-Erlang blocks, one of length $b_1 = 1$ with rate λ_1, and one of length $b_2 = 3$ with rate λ_2 and feedback probability z_2. The CTMC may be entered at any state, and therefore at any block, but can only reach absorption by passing through the remaining blocks. Furthermore, the CTMC may go back to previous states, however, it cannot go back to previous FE blocks. The importance of FE-diagonal structures lies in the fact that the sub-generator of such a structure can have complex eigenvalues, which is not the case with the bi-diagonal form, which is a special case of the FE-diagonal form, with lengths $b_j = 1$ and feedback probabilities $z_j = 0$.

2.3.2.2 Branch Structures

In this section we describe three important branch structures. The first definition describes the structure of the CTMC for Hyper-Erlang distributions, while the second extends the concept to mixtures of Feedback-Erlang distributions. The third definition generalises the second definition even further, by allowing chains of FE distributions in the blocks.



(a) Hyper-Erlang distribution (b) Hyper-Feedback-Erlang distribution

Figure 2.5: CTMCs of Hyper-Erlang and Hyper-Feedback-Erlang distributions with two branches.

Definition 2.3.6 (Hyper-Erlang representation [TBT06])**.** *Let* $(\boldsymbol{\alpha}, \mathbf{Q})$ *be a Markovian representation in branch structure. The representation is in* Hyper-Erlang (HErD) form *if, for* $k = 1, \ldots, m$*:*

$$\alpha_i > 0 \quad \textit{for} \quad i \in \left\{1, 1+b_1, \ldots, 1+\sum_{k=0}^{m-1} b_k\right\}, \tag{2.40}$$

$$\alpha_i = 0 \quad \textit{else} \tag{2.41}$$

and

$$\mathbf{Q}_k \in \mathbb{R}^{b_k \times b_k} \tag{2.42}$$

$$\mathbf{Q}_k = \begin{pmatrix} -\lambda_k & \lambda_k & & \\ & \ddots & \ddots & \\ & & & \\ & & -\lambda_k & \lambda_k \\ & & & -\lambda_k \end{pmatrix}, \tag{2.43}$$

with $\lambda_k > 0$*. That is, the representation consists of* m *Erlang branches with initial probabilities* $\alpha_1, \alpha_{1+b_1}, \ldots, \alpha_{1+\sum_{k=1}^{m-1} b_k}$*, lengths* $b_1, \ldots, b_m$ *and rates* $\lambda_1, \ldots, \lambda_m$*. Equivalently, we can use the notation*

$$\mathbf{H} = (\boldsymbol{\beta}, (b_1, \lambda_1), \ldots, (b_m, \lambda_m)), \tag{2.44}$$

where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_m) \in \mathbb{R}^m$ *gives the initial probabilities of the branches:*

$$\beta_1 = \alpha_1, \beta_2 = \alpha_{1+b_1}, \ldots, \beta_m = \alpha_{1+\sum_{k=1}^{m-1} b_k}, \tag{2.45}$$

and the branches are defined by the tuples $(b_k, \lambda_k), k = 1, \ldots, m$*, where* b_k *gives the length and* λ_k *is the rate of the* k*th branch.*

An example of the general structure of the CTMC of a Hyper-Erlang distribution is shown in Figure 2.5(a). The distribution shown here has $m = 2$ branches of length $b_1 = 4$ and $b_2 = 3$, respectively. The initial probabilities and the transition rates are given by $\boldsymbol{\beta} = (\beta_1, \beta_2)$ and λ_1, λ_2. The size of this representation is $n = b_1 + b_2 = 7$. Each branch can only be entered at its first state and then has to be traversed completely before absorption. Furthermore, the CTMC cannot change to a state that belongs to another branch. Common examples for Hyper-Erlang distributions are the Erlang distribution, which has one branch and initial probability vector $\boldsymbol{\beta} = (1, 0, \ldots, 0))$, and the Hyper-Exponential distribution, where all branches have length 1.

As with the bi-diagonal representations, the Hyper-Erlang structures can be generalised by using the FE-diagonal structure for the block matrices.

Definition 2.3.7 (Hyper-Feedback-Erlang representation)**.** *Let* $(\boldsymbol{\alpha}, \mathbf{Q})$ *be a Markovian representation in branch structure. The representation is in* Hyper-Feedback-Erlang (HFED) form *if all* $\mathbf{Q}_k, k = 1, \ldots, m$ *are in Feedback-Erlang form. We will use the notation* $(\boldsymbol{\alpha}, \boldsymbol{\Upsilon})$*, where*

$$\boldsymbol{\Upsilon} \quad = \quad ((b_1, \lambda_1, z_1), \ldots, (b_m, \lambda_m, z_m)). \tag{2.46}$$

We use the notation $\hat{\boldsymbol{\alpha}}^{(k)}$ *to refer to the initial probability vector of the* k*th Feedback-Erlang block:*

$$\hat{\boldsymbol{\alpha}}^{(k)} \quad = \quad (\alpha_{i_k}, \alpha_{i_k+1}, \ldots, \alpha_{i_k+b_k}) \in \mathbb{R}^{b_k}, \tag{2.47}$$

where

$$i_k \quad = \quad \sum_{l=1}^{k-1} b_l + 1 \tag{2.48}$$

is the index of the first state of the branch.

Figure 2.5(b) shows the CTMC of a distribution in Hyper-Feedback-Erlang form. Observe that in each branch there is a Feedback-Erlang loop (with feedback probabilities z_1, z_2), and that the branches can be entered at any state.

In [HT11] Horváth and Telek give a different definition for the Hyper-Feedback-Erlang distribution. Instead of only allowing a single FE block in each branch, this definition allows chains of multiple identical FE blocks. In order to keep the analogy between the Hyper-Erlang and Hyper-Feedback-Erlang distributions as mixtures of the respective distributions, throughout this work we use the term Hyper-FE distribution only for distributions fulfilling Definition 2.3.7, and refer to the definition in [HT11] by the name Extended Hyper-Feedback-Erlang (eHFED) distribution:

Definition 2.3.8 (Extended Hyper-Feedback-Erlang representation [HT11])**.** *Let* $(\boldsymbol{\alpha}, \mathbf{Q})$ *be a Markovian representation in branch structure. The representation is in* Extended Hyper-Feedback-Erlang (eHFED) form *if all* $\mathbf{Q}_k, k = 1, \ldots, m$ *in Definition 2.3.2 are of the form*

$$\mathbf{Q}_k = \begin{pmatrix} \mathbf{F}_k & \mathbf{F}_k \mathbb{1}\mathbf{e}_1 & & \\ & \ddots & \ddots & \\ & & & \mathbf{F}_k \mathbb{1}\mathbf{e}_1 \\ & & & \mathbf{F}_k \end{pmatrix} \in \mathbb{R}^{m_k b_k \times m_k b_k}, \tag{2.49}$$

where $\mathbf{F}_k \in \mathbb{R}^{b_k \times b_k}$ *is the subgenerator matrix of the* k*th FE block with size* b_k*, and* $m_k \geq 1$ *denotes the multiplicity of this FE block. We denote the* k*th branch of an eHFED distribution by the vector*

$$(m_k, b_k, \lambda_k, z_k), \tag{2.50}$$

where b_k*,* λ_k*, and* z_k *define an FE block. An extended Hyper-FE representation with* m *branches is then given by* $(\boldsymbol{\alpha}, \boldsymbol{\Upsilon})$*, where*

$$\boldsymbol{\Upsilon} = ((m_1, b_1, \lambda_1, z_1), \ldots, (m_m, b_m, \lambda_m, z_m)) \tag{2.51}$$

denotes the FE-diagonal branches. The initial probability vector of the k*th branch is given by*

$$\hat{\boldsymbol{\alpha}}^{(k)} = (\alpha_{i_k}, \alpha_{i_k+1}, \ldots, \alpha_{i_k+b_k}) \in \mathbb{R}^{m_k b_k}, \tag{2.52}$$

where

$$i_k = \sum_{l=1}^{k-1} b_l + 1 \tag{2.53}$$

is the index of the first state of the branch. Augmenting our definition for FE-diagonal representations, we use the notation $\hat{\boldsymbol{\alpha}}^{(k,j)}$ *to refer to the initial probabilities*

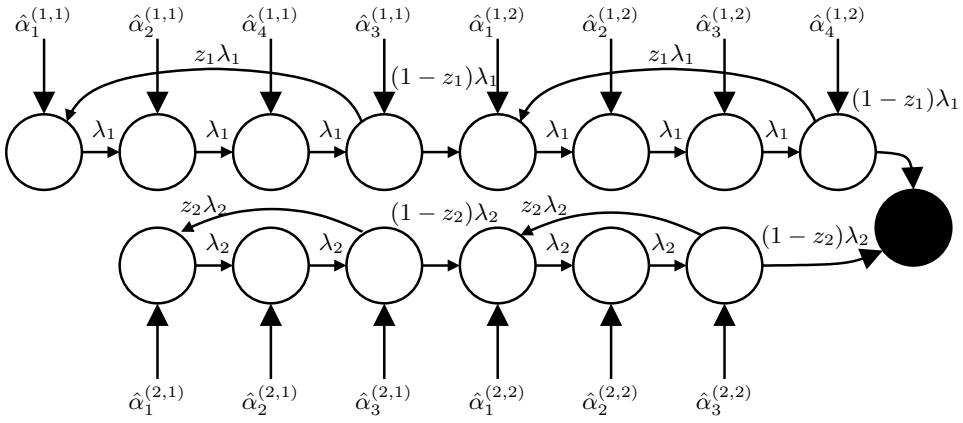


Figure 2.6: Extended Hyper-Feedback-Erlang distribution.

of the jth FE block in the kth branch:

$$\hat{\boldsymbol{\alpha}}^{(k)} = \left(\hat{\boldsymbol{\alpha}}^{(k,1)}, \ldots, \hat{\boldsymbol{\alpha}}^{(k,m_k)}\right), \tag{2.54}$$

with

$$\hat{\boldsymbol{\alpha}}^{(k,j)} = \left(\alpha_{i_k+(j-1)b_k}, \alpha_{i_k+(j-1)b_k+1}, \ldots, \alpha_{i_k+jb_k-1}\right). \tag{2.55}$$

The CTMC of a distribution in extended Hyper-Feedback-Erlang form with two branches is shown in Figure 2.6. In contrast to the Hyper-Feedback-Erlang form, each branch may contain multiple FE blocks in chain structure. All FE blocks within a branch have the same length, rate, and feedback probability.

Note that the Hyper-FE and Extended-Hyper-FE representations differ with respect to the phase-type distributions that they can represent. The Hyper-FE distribution can only represent distributions where all poles of the Laplace-Stieltjes Transform of the distribution function have multiplicity one. In contrast, the eHFED representation exists for all PH distributions, as shown in [HT11]: First, a sub-generator matrix in eHFED form that expresses all eigenvalues of the original representation by chains of identical FE blocks (with chain lengths corresponding to the multiplicity of the eigenvalues) is constructed. The initial vector is then computed by a generalised similarity transformation between the two representations. While this representation does exist for every PH distribution, it is not guaranteed to be Markovian.

2.4 Canonical Representations

While in general representations for phase-type distributions are not unique, several canonical representations have been proposed. In order for a representation of the PH class or one of its sub-classes to be canonical it must be possible to transform every distribution within that class to this representation in a unique way. That is, there must not be another canonical representation of the same distribution of the same size, but with different parameters. This characteristic enables the use of canonical representations to determine whether two representations $(\boldsymbol{\alpha}, \mathbf{Q})$, $(\boldsymbol{\alpha}', \mathbf{Q}')$ define the same distribution. Second, canonical representations aim to reduce the number of parameters and to simplify the structure of the representation, thus allowing more efficient fitting, analysis, and simulation methods. In the following we discuss Cumani's Canonical Form 1 (CF-1) [Cum82] and the Monocyclic form introduced in [MC99], as these are the most common ones.

2.4.1 The Canonical Form 1 (CF-1) for Acyclic Phase-type Distributions

In [Cum82] Cumani proposed the following bi-diagonal representation as a canonical form for all acyclic phase-type distributions:

Definition 2.4.1 (CF-1 form [Cum82])**.** *The* Canonical Form 1 (CF-1 form) *is a Markovian bi-diagonal representation* $(\boldsymbol{\alpha}, \boldsymbol{\Lambda})$ *where the elements of the diagonal, given in the vector* $\boldsymbol{\Lambda}$*, are in increasing order:*

$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n.$$

According to the following theorem, the CF-1 form is indeed a canonical representation:

Theorem 2.4.1 (Canonicity of the CF-1 form, [Cum82, O'C91])**.** *Let* $F(t)$ *be an acyclic phase-type distribution with Markovian representation* $(\boldsymbol{\alpha}, \mathbf{Q})$ *of size* n*. Then, there exists a unique CF-1 representation* $(\boldsymbol{\alpha}', \mathbf{Q}')$ *of the same size that defines the same distribution.*

Proofs for Theorem 2.4.1 may be found in [Cum82] and in [O'C91]. Cumani's proof is based on the observation that in an acyclic CTMC there is only a finite number of paths to absorption, and that each of these paths is of finite length less than or equal to the size of the distribution. With adjustments to the initialisation vector, the distribution can be expressed as a weighted sum of these paths. O'Cinneide, on the other hand, applies a geometric approach that uses the concept of invariant polytopes.

The CF-1 form for an APH given as $(\boldsymbol{\alpha}, \mathbf{Q})$ can be obtained by a similarity transformation. A procedure for constructing the similarity transformation matrix is given in [HZ06]. Theorem 2.4.1 guarantees the existence of a CF-1 representation of the same size. Smaller CF-1 representations may exist if there is redundancy in the original representation [Neu81, HZ06, Pul09].

An important property of the CF-1 form is that the transformation of an APH to CF-1 considerably reduces the number of parameters: A general APH representation has n initial probabilities $\alpha_1, \ldots, \alpha_n$ and n^2 entries in the subgenerator matrix $\mathbf{Q}$, i.e. the number of parameters is $n + n^2$. Even with the upper-triangular form the number of parameters is still $n + 1/2(n+1)n$. In the CF-1 form $\mathbf{Q}$ is an upper bi-diagonal matrix with n entries $\lambda_{1,1}, \ldots, \lambda_{n,n}$ and $\lambda_{i,i} = -\lambda_{i+1,i}$. The CF-1 form therefore has $2n$ parameters.

2.4.2 The Monocyclic Form for General PH Distributions

General PH distributions may have complex poles, and the poles of a PH distribution are given by the eigenvalues of the subgenerator matrix $\mathbf{Q}$. As the eigenvalues of a bi-diagonal representation $(\boldsymbol{\alpha}, \boldsymbol{\Lambda})$ are equal to the entries of the diagonal, and $\mathbf{Q} \in \mathbb{R}^{n \times n}$, it is easy to see that a bi-diagonal structure like the CF-1 form cannot represent phase-type distributions with complex poles.

For this reason, [MC99] proposed the Monocyclic form and the Markovian Monocyclic form. We re-state the definition here in terms of the FE-diagonal form introduced before:

Definition 2.4.2 (Monocyclic and Markovian Monocyclic form [MC99])**.** *The* Monocyclic form *is a Feedback-Erlang-diagonal representation* $(\boldsymbol{\alpha}, \boldsymbol{\Upsilon})$ *where the Feedback-Erlang blocks are positioned along the diagonal in increasing order of the absolute value of the dominant eigenvalues* $\rho_i, i = 1, \ldots, m$ *of the matrices* $\mathbf{F}_i$ *corresponding to the FE blocks:*

$$|\rho_1| \leq |\rho_2| \leq \cdots \leq |\rho_m|$$

The representation is said to be in Markovian Monocyclic form *if the vector* $\boldsymbol{\alpha}$ *is Markovian, i.e.* $\boldsymbol{\alpha} \geq \mathbf{0}$ *and* $\boldsymbol{\alpha}\mathbf{1}\mathbf{I} = 1$.

As summarised in Lemma 2.3.4, a Feedback-Erlang block of order $b > 2$ and with feedback probability $z > 0$ has at least one conjugate-complex pair of eigenvalues [MC99]. Therefore, a chain of FE blocks can be used to represent the complex eigenvalue pairs of a general phase-type distribution. Note that if $z_i = 0$ for all FE blocks $i = 1, \ldots, b$ the Monocyclic form is equivalent to the CF-1 form. That is, the CF-1 form is a special case of the Monocyclic form.

The following theorem combines Proposition 2 and Theorem 2 of [MC99]:

Theorem 2.4.2 (Markovian Monocyclic form, [MC99])**.** *Let $F(t)$ be a phase-type distribution with representation $(\boldsymbol{\alpha}, \mathbf{Q})$. Then, there exists a Markovian Monocyclic representation $(\boldsymbol{\beta}, \mathbf{M})$ of the same distribution.*

Proposition 2 in [MC99] shows that every PH distribution $F(t)$ has a Monocyclic representation, and gives the following method for its construction: Let $(\boldsymbol{\alpha}, \mathbf{Q})$ be a representation of $F(t)$ of size n, let $\rho_1, \ldots, \rho_n$ be the eigenvalues of $\mathbf{Q}$, ordered such that the elements of complex-conjugate pairs are immediately adjacent, i.e.

$$\mathrm{Im}(\rho_i) > 0 \Leftrightarrow \mathrm{Im}(\rho_{i+1}) = -\mathrm{Im}(\rho_i),$$

and let m be the number of eigenvalues with distinct real part. Then, a Monocyclic representation $(\boldsymbol{\alpha}', \mathbf{Q}')$ can be obtained by constructing Feedback-Erlang blocks $\mathbf{F}_1, \ldots, \mathbf{F}_m$ such that each FE block represents either a real eigenvalue ρ_i or a complex conjugate pair of eigenvalues ρ_i, ρ_{i+1}. The sub-generator $\mathbf{Q}'$ is then defined as

$$\mathbf{Q}' = \begin{pmatrix} \mathbf{F}_1 & -\mathbf{F}_1 \mathbb{1}\mathbf{e}_1 & & \\ & \ddots & \ddots & \\ & & \mathbf{F}_{m-1} & -\mathbf{F}_{m-1}\mathbb{1}\mathbf{e}_1 \\ & & & \mathbf{F}_m \end{pmatrix} \in \mathbb{R}^{n' \times n'}, \tag{2.56}$$

where the FE blocks are ordered by the dominant eigenvalue of their matrices, $\mathbf{F}_i$, and $n' = \sum_{i=1}^{m} m_i$ is the size of the representation. In a second step, a new initialisation vector $\boldsymbol{\alpha}'$ is computed such that $(\boldsymbol{\alpha}', \mathbf{Q}')$ represents $F(t)$. The new vector can be computed as follows: As we have n' unknowns $\alpha_1, \ldots, \alpha_{n'}$, we must construct a system of n' equations,

$$F_{(\boldsymbol{\alpha}', \mathbf{Q}')}(t_1) = F_{(\boldsymbol{\alpha}, \mathbf{Q})}(t_1) \tag{2.57}$$

$$\vdots \tag{2.58}$$

$$F_{(\boldsymbol{\alpha}', \mathbf{Q}')}(t_{n'}) = F_{(\boldsymbol{\alpha}, \mathbf{Q})}(t_{n'}), \tag{2.59}$$

and solve it for $\boldsymbol{\alpha}'$.

In general, the initialisation vector obtained by this method is not Markovian. Theorem 2 in [MC99] then states that a Markovian initialisation vector can be found by adding phases to the Monocyclic generator. The original proof given in [MC99] relies on the invariant polytopes approach by [O'C91]. This method provides an elegant way of proving the statement using a geometric argument. On the other hand, this elegant approach comes at the cost of requiring a quite complex set of non-standard mathematical tools.

A more accessible proof can be sketched as follows: Let $(\boldsymbol{\alpha}, \mathbf{Q})$ be a non-Markovian Monocyclic representation and let $(\boldsymbol{\beta}, \mathbf{M})$ be a representation where an

Erlang distribution of length k with rate λ has been appended to the last state of the original representation, i.e.

$$\mathbf{M} := \begin{pmatrix} \mathbf{Q} & -\mathbf{Q}\mathbb{1}\mathbf{e}_1 & & & \\ & -\lambda & \lambda & & \\ & & \ddots & \ddots & \\ & & & \ddots & \lambda \\ & & & & -\lambda \end{pmatrix} \in \mathbb{R}^{(n+k)\times(n+k)} \tag{2.60}$$

and $\boldsymbol{\beta} \in \mathbb{R}^{n+k}$. The operation of adding k phases with rate λ is described by the generalised similarity transformation matrix

$$\hat{\mathbf{W}} := \left(\mathbf{W}^k | \mathbf{W}^{k-1}\boldsymbol{\omega} | \ldots | \mathbf{W}^0\boldsymbol{\omega}\right) \in \mathbb{R}^{n\times(n+k)} \text{ with} \tag{2.61}$$

$$\mathbf{W} := \left(\mathbf{I} + \frac{1}{\lambda}\mathbf{Q}\right) \tag{2.62}$$

$$\boldsymbol{\omega} := \mathbb{1}\mathbf{I} - \mathbf{W}\mathbb{1}\mathbf{I}. \tag{2.63}$$

$\hat{\mathbf{W}}$ is an $n\times(n+k)$ matrix consisting of the square matrix $\mathbf{W}^k$ and k column vectors $\mathbf{W}^{k-1}\boldsymbol{\omega}, \ldots, \mathbf{W}^0\boldsymbol{\omega}$. It can be easily shown that

$$\hat{\mathbf{W}}\mathbf{M} = \mathbf{Q}\hat{\mathbf{W}} \tag{2.64}$$

$$\hat{\mathbf{W}}\mathbb{1}\mathbf{I} = \mathbb{1}\mathbf{I}. \tag{2.65}$$

Because $\hat{\mathbf{W}}$ is a generalised similarity transformation matrix, $(\boldsymbol{\beta}, \mathbf{M})$ with $\boldsymbol{\beta} = \boldsymbol{\alpha}\hat{\mathbf{W}}$ describes the same distribution as $(\boldsymbol{\alpha}, \mathbf{Q})$. One can then show that by adding a sufficiently high number of phases the entries of the initial vector that correspond to the first n states of the distribution become non-negative, and that a suitable choice of λ ensures that the k additional entries of the new initial vector are also non-negative. Both insights follow by contradiction from the precondition that $f(t) > 0$ for $t > 0$. Therefore, there exist k, λ such that $\boldsymbol{\beta} = \boldsymbol{\alpha}\hat{\mathbf{W}} \geq \mathbf{0}$. The following example illustrates the approach.

Example 2.4.1 (Markovian Monocyclic Representation)**.** *Consider the Monocyclic representation* $(\boldsymbol{\alpha}, \mathbf{Q})$ *with*

$$\boldsymbol{\alpha} = (1.1, -0.3, 0.2, 0) \quad \textit{and} \quad \mathbf{Q} = \begin{pmatrix} -1 & 1 & & \\ & -2 & 2 & \\ & & -2 & 2 \\ & 0.1 & & -2 \end{pmatrix}. \tag{2.66}$$

This representation is non-Markovian. We define $\lambda := 5$ *and obtain*

$$\mathbf{W} := \mathbf{I} + \frac{\mathbf{Q}}{5} = \begin{pmatrix} 0.8 & 0.2 & & \\ & 0.6 & 0.4 & \\ & & 0.6 & 0.4 \\ & 0.02 & & 0.6 \end{pmatrix} \tag{2.67}$$

$$\boldsymbol{\omega} := \mathbb{1}\mathbf{I} - \mathbf{W}\mathbb{1}\mathbf{I} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0.38 \end{pmatrix}. \tag{2.68}$$

For $k = 1$ *we have the generalised similarity transformation matrix*

$$\hat{\mathbf{W}} = \begin{pmatrix} \mathbf{W} & |\boldsymbol{\omega} \end{pmatrix} = \begin{pmatrix} 0.8 & 0.2 & & & \\ & 0.6 & 0.4 & & \\ & & 0.6 & 0.4 & \\ & 0.02 & & 0.6 & 0.38 \end{pmatrix} \tag{2.69}$$

$$\tag{2.70}$$

By applying $\hat{\mathbf{W}}$ *to* $(\boldsymbol{\alpha}, \mathbf{Q})$ *we obtain the Markovian Monocyclic representation*

$$\boldsymbol{\beta} = (0.88, 0.04, 0, 0.08, 0) \quad \textit{and} \quad \mathbf{M} = \begin{pmatrix} -1 & 1 & & & \\ & -2 & 2 & & \\ & & -2 & 2 & \\ & 0.1 & & -2 & 1.9 \\ & & & & -5 \end{pmatrix}. \tag{2.71}$$

Note that by different choice of λ *and* k *we can obtain other Markovian Monocyclic representations, e.g. for* $\lambda = 6$ *and* $k = 2$ *we get*

$$\begin{aligned} \boldsymbol{\beta}' &= (0.763889, 0.142778, 0.0166667, 0.0555556, 0.021111, 0) \textit{ and} \\ \mathbf{M}' &= \begin{pmatrix} -1 & 1 & & & & \\ & -2 & 2 & & & \\ & & -2 & 2 & & \\ & 0.1 & & -2 & 1.9 & \\ & & & & -6 & 6 \\ & & & & & -6 \end{pmatrix}. \end{aligned} \tag{2.72}$$

Example 2.4.1 illustrated the transformation of a non-Markovian Monocyclic representation to a Markovian Monocyclic representation. The example also served to show that this transformation is not unique, since different Markovian representations can be constructed depending on the choice of k and λ. While this means

that the Markovian Monocyclic representation is not canonical in the strict sense of the term, one might easily add additional criteria, such as minimal size or minimal size of the added rate to enforce uniqueness, if this is required. Throughout this work we will consider the Markovian Monocyclic representations canonical, in the sense that every PH distribution has such a representation. As we do not consider non-Markovian canonical representations, we will in general simplify terminology by using the term *Monocyclic form* to refer to Markovian Monocyclic representations only.

2.5 Concluding Remarks

In this chapter we summarised the theoretical background on phase-type distributions required for the remainder of this work. We introduced special representations and sub-classes of PH distributions and discussed canonical representations for the acyclic and general PH class. Special representations will play a major role in both random-variate generation (Chapter 3) and in phase-type fitting (Chapter 5). We also defined similarity transformation for switching between different representations of the same PH distribution. These operations are applied extensively in the optimisation approaches studied in Chapter 4.

Chapter Three

Algorithms for Random-Variate Generation from Phase-type Distributions

Random variates are required to represent stochastic behaviour in simulation and testbeds. Although phase-type distributions are well-known for their desirable properties in fitting empirical data and in analytical approaches, their use for random-variate generation is still a relatively unexplored area, both from a practical and from a theoretical perspective. With the exception of simple sub-classes such as the Erlang or hyper-exponential distributions, phase-type distributions are generally neither discussed in textbooks on simulation (see e.g. [Jai91, Law06]), nor supported in simulation environments. A few algorithms have been proposed in [NP81, BPdL98, HT11], which use different approaches and in some cases support larger classes of distributions that include the PH class. These algorithms address the cost of random-variate generation to some extent, but do not fully exploit the structural properties for optimisation of the computational costs. From a theoretical perspective, the existing body of work appears quite scattered, with little to no overview of approaches or comparative discussion of the costs of random-variate generation. Furthermore, existing algorithms are not amenable to reducing the cost by suitable choice of a PH representation. In this chapter we provide a survey of algorithms for random-variate generation from PH distributions, especially taking into account the costs of the algorithms in terms of operations that we identify as atomic. Our focus in this work is on methods that use the structure of the Markov chain. We propose two new algorithms for random-variate generation that rely on canonical forms. These algorithms are more efficient than generic algorithms; their primary advantage, however, lies in the fact that they enable elegant expressions for the cost metrics. Such expressions are required for optimisation of representations for efficient random-variate generation.

This chapter is structured as follows: We first recall the classification scheme for random-variate algorithms introduced in [Jai91], which we will use to provide a structure to our discussion of the random-variate generation algorithms. In Section 3.2 we describe atomic operations used in the algorithms and define cost metrics based on these operations. We then describe existing algorithms and discuss their costs. We introduce two new algorithms and derive elegant expressions for the costs of these algorithms. Finally, we provide a measurement study on the computational costs incurred by the atomic operations in typical implementations.

3.1 General Terminology and Structure of the Survey of Algorithms

Throughout this work we follow [Jai91] in the distinction between *random variates* and *random numbers*: *Random variates* may follow any distribution (or indeed any stochastic process), while *random numbers* only refer to the uniform distribution on $(0, 1)$. We are concerned purely with the generation of random variates and assume that random numbers are given.

We apply the following classification of algorithms for random-variate generation from [Jai91]: *Inversion methods* operate by inverting the distribution function. Inversion methods can be very efficient, but require an explicit expression for the inverse of the distribution function. *Acceptance/Rejection methods* draw random variates from a distribution whose density majorises the target distribution and accept only those samples that belong to the target distribution. *Composition methods* draw a random variate from a mixture of distributions by first randomly selecting one of the distributions and then generating a random variate of that distribution. These methods are applicable if there exist efficient methods for the component distributions. Similarly, *convolution methods* generate random variates from a distribution which is a convolution of simpler distributions. Both composition and convolution methods are special cases of *characterisation methods*. Characterisation methods rely on a characterisation of the target distribution in terms of distributions for which efficient methods exist. Finally, *empirical methods* draw random variates from an empirical distribution represented by measurements.

Both the inversion and the acceptance/rejection method have been proposed for PH-random-variate generation; the main focus, however, lies on algorithms that can be subsumed under the label of characterisation methods, as they utilise the structure of the Markov chain. Methods that draw from the empirical distribution are not relevant with respect to generating PH-distributed random variates.

3.2 Atomic Operations and Cost Metrics

The algorithms discussed in the following can be broken down to three atomic operations, which allow us to define cost metrics that enable comparison between the algorithms. These operations were chosen based on their common use in the algorithms and their availability as basic operations in implementation. We first describe common operations in random-variate generation. We then identify atomic operations and introduce three cost metrics for PH-random-variate generation.

Generation of a uniform random number in $(0, 1)$**.** Uniform random numbers are required in all algorithms that generate random variates. In the context of PH distributions, random numbers are used as samples in the inversion method, for determining acceptance in the acceptance/rejection method, and for state selection and sojourn-time computation in the characterisation methods.

There exists a large body of literature on random-number generation for simulation (e.g. [Jai91, Law06]), and in the context where random variates are required, e.g. in simulation environments and operating systems, routines for random-number generation are typically provided. With simulation environments in particular, one should use the random numbers they provide, as these are suited to simulation and provide e.g. independent random-number streams. We do not explicitly distinguish between true randomness and pseudo-randomness of the random numbers, as this distinction has no bearing on the random-variate generation algorithms we discuss. Whether these are truly random or pseudo-random largely depends on the nature of the random numbers that are used. While in simulation one often prefers pseudo-randomness due to repeatability of experiments (cf. [Jai91]), true random variates may be useful in other contexts.

We consider the drawing of a sample from the uniform distribution on $(0, 1)$ an atomic operation with a given cost. We use U to denote a uniform random variable, and u to describe a sample from this variable. If several samples are required, we denote them by sub-scripts, e.g. $u_i, u_{i+1}, \ldots$.

Generation of a random variate with exponential distribution. Characterisation methods for phase-type distributions utilise the fact that every PH distribution can be represented by a Continuous-Time Markov Chain (CTMC) to generate random variates by 'playing' this CTMC until the absorbing state is reached. Playing the CTMC involves drawing state sojourn times for each traversed state. As state sojourn-times are exponentially distributed, drawing a sample from the exponential distribution is a common operation in these methods.

Using the inversion method (see also Section 3.3), a sample from the exponential distribution with rate λ can be computed as

$$\mathrm{Exp}(\lambda) \quad := \quad -\frac{1}{\lambda} \ln u, \tag{3.1}$$

where u is a uniform random number [Jai91, Hav98]. Generation of an exponential sample thus requires one logarithm and one uniform random number.

Generation of an Erlang-distributed random variate. In characterisation methods, Erlang-distributed random variates can be used to describe the sojourn times for chains of states with identical exponential distributions. Since the Erlang distribution of length b with rate λ is defined as the sum of b independent exponential random variates with rate λ, drawing a sample from the Erlang distribution is equivalent to drawing and adding b samples from the exponential distribution with rate λ. The importance of the Erlang distribution lies in the fact that an Erlang-distributed sample $\mathrm{Erl}(\lambda)$ requires only one logarithm operation, as opposed to b logarithm operations for the sum of exponentials [NP81, Hav98]:

$$\mathrm{Erl}(\lambda, b) \quad := \quad \sum_{i=1}^{b} -\frac{1}{\lambda} \ln(u_i) \tag{3.2}$$

$$= \quad -\frac{1}{\lambda} \ln \left(\prod_{i=1}^{b} u_i \right). \tag{3.3}$$

An Erlang-distributed sample generated using (3.3) needs b uniform random numbers $u_1, \ldots, u_b$ and 1 logarithm operation.

We note in passing that (3.3) can incur serious numerical problems in actual implementation: For large b, a long series of floating-point numbers $u_1, \ldots, u_b$ with $0 < u_i < 1$ is multiplied. Because this product approaches zero, it may exceed the range of numbers that can be represented by a floating-point number. In this case, the computed product is zero, for which the logarithm is not defined. The problem could be addressed by e.g. splitting the sequence of random numbers into $N - 1$ segments of length $b' = \lfloor \frac{b}{N} \rfloor$ and one segment of length $b'' = b - Nb'$ such that no underflow occurs in the products corresponding to each segment, i.e.

$$\mathrm{Erl}(\lambda, b) \quad = \quad -\frac{1}{\lambda} \left(\sum_{j=0}^{N-2} \ln \left(\prod_{i=jb'+1}^{(j+1)b'} u_i \right) + \ln \left(\prod_{(N-1)b'+1}^{b''} u_i \right) \right). \tag{3.4}$$

Erlang-distributed random variates generated using (3.4) require N logarithm operations. In the theoretical discussion of the algorithms we assume that (3.3) is used.

Generation of a geometric random variate. The geometric distribution describes the number of trials until an event occurs. In some characterisation methods, this distribution is used to compute the number of times a loop in the Markov chain is traversed. A sample
textGeo(p) from the geometric distribution (starting from 0) with success probability p for the event can be drawn by the inversion method [Jai91, Law06]:

$$\mathrm{Geo}(p) \quad = \quad \left\lfloor \frac{\ln(u)}{\ln(1-p)} \right\rfloor, \tag{3.5}$$

requiring 1 uniform random number and 2 logarithms. Note that if the denominator in (3.5) is constant, it may be pre-computed (as suggested in [HT11]). However, the routines that use the geometric distribution typically require random variates from several different geometric distributions, and thus for each geometric random variate a lookup for the appropriate constant term would be needed. This lookup operation would replace the logarithm operation in (3.5) and also incur some computational cost. In order to provide simple expressions, throughout the following we assume that no such optimisation takes place.

Computation of the matrix exponential. Inversion methods for random-variate generation from PH distributions require the computation of the matrix exponential. There exists a large body of work on methods for computing the matrix exponential (see [ML03]). As the costs of using these methods depend not only on the method, but also on the size of the given matrix, the matrix exponential is not a suitable operation for cost comparisons and a simpler operation. We use the *scalar* exponential as another common operation. As described in [Ber09], p. 711f, the exponential of a matrix of size n can be expressed using linear combinations of n scalar exponentials, and thus the scalar exponential is an appropriate operation.

3.2.1 Atomic Operations

Random-variate generation from PH distributions requires uniform random numbers, samples from the exponential, Erlang, and geometric distributions, and computation of the matrix exponential. For the purposes of this work, we treat the generation of uniform random numbers, the computation of a logarithm, and the computation of a scalar exponential as atomic operations. That is, we assume that these operations are provided by the environment and incur some constant computational cost. While it may be possible to split them still further, this would be of little value for discussing the efficiency of different algorithms for random-variate generation in relation to each other, since the costs would then also depend on the particular implementations of the operations.

3.2.2 Cost Metrics

Given the above three atomic operations, the following three cost metrics appear useful for our discussion of the costs of random-variate algorithms:

Definition 3.2.1 (Cost Metrics for PH-Random-Variate Generation)**.** *Let* #uni *be the* number of uniform random variates *that need to be generated, let* #ln *be the* number of logarithm operations *that must be performed, and let* #exp *be the* number of scalar exponentials *that must be computed for generating one PH-distributed random variate from a given PH distribution with representation* $(\boldsymbol{\alpha}, \mathbf{Q})$ *of size* n*.*

Using these metrics, we can compare the costs that the algorithms incur when generating random variates from a given PH distribution with a given representation. We consider both worst-case and average costs. In general, these costs differ. The former specifies the highest number of atomic operations performed, and thus the largest amount of computation time required by the algorithm. The latter describes the average number of operations and the average amount of computation time.

3.3 The Inversion Method for Phase-type Random-Variate Generation

Inversion methods are based on the observation that the values u of the distribution function $F(t)$ follow a uniform distribution on $[0, 1]$. Consequently, given samples u from the uniform distribution,

$$t \;=\; F^{-1}(u) \tag{3.6}$$

is distributed according to $F(t)$ [Jai91]. If an efficient way for computing the inverse of $F(t)$ is known, inversion methods can be very efficient routines for random-variate generation, since they require only a single uniform random number. For instance, a random variate from the exponential distribution with rate λ and distribution function $F(t) = 1 - \mathrm{e}^{-\lambda t}$ can be obtained by the expression given in (3.1).

The exponential distribution is the simplest non-trivial phase-type distribution, and exponentially-distributed random variates play an important role in characterisation methods. However, the analytical inversion approach does not generalise to phase-type distributions with size $n > 1$, since the matrix and vector structures in the distribution function,

$$F(t) \;=\; 1 - \boldsymbol{\alpha}\, \mathrm{e}^{\mathbf{Q}t}\, \mathbb{1}$$

do not admit an explicit expression of $F^{-1}(t)$.

Brown et al. [BPdL98] propose application of inversion methods for random-variate generation from matrix-exponential (ME) distributions based on numerical

inversion of the distribution function. Since PH distributions are a subclass of ME distributions, this approach is also applicable for PH distributions. The following pseudo-code summarises the algorithm given in [BPdL98]:

Procedure `NumericalInversion`$(\boldsymbol{\alpha}, \mathbf{Q})$:
Draw a sample u from the uniform distribution on $(0, 1)$.
$a := 0, b := t_{max}$
$t := \frac{a+b}{2}$
while $|F(t) - u| > \epsilon$ **do**
  if $F(t) < u$ **then**
    $a := t$
  else
    $b := t$
  end if
  $t := \frac{a+b}{2}$
end while
return c

The algorithm works as follows: First, a uniform random number is generated, and then t such that $F(t) \approx u$ is identified by binary search. The binary search starts with the interval $[0, t_{max}]$ and iteratively shrinks the interval until the center of the interval is sufficiently close to u. Each step of the search for t requires computing the value of $F(t)$, which involves computing the matrix exponential $e^{\mathbf{Q}t}$. The upper limit t_{max} depends on the application and on the method used for computation of the matrix exponential. Brown et al. propose the scaling-and-squaring method (cf. [ML03]) to compute $e^{\mathbf{Q}t}$ and note that this method is only accurate for $u < t_{max} < 1$. Random numbers u close to 1 correspond to the tail of the distribution. For $u \geq t_{max}$, Brown et al. use an exponential approximation for the tail. We omit these steps here, as they are specific to the scaling-and-squaring method.

Costs of numerical inversion. Using our metrics, and assuming that the matrix exponential is computed using scalar exponentials, costs for the `NumericalInversion` method are measured by the number of uniform random numbers and by the number of scalar exponentials (no logarithm operations are performed). The algorithm requires exactly one uniform random number. The worst case depends on how many steps are taken by the binary search in the worst case, i.e. $\log(1/\delta)$ steps for an accuracy of δ. The number of steps depends on the accuracy δ. As each step involves n scalar exponentials, worst-case costs can be expressed as

$$\#exp_{\text{WC, NumericalInversion}} = n \log(1/\delta), \tag{3.7}$$

e.g. $\#exp = 19n$ for an accuracy of $\delta = 10^{-6}$, as considered by Brown et al. in [BPdL98]. For numerical inversion using binary search, worst case and average

costs are equal.

3.4 The Acceptance-Rejection Method for Random-Variate Generation

An acceptance/rejection method for generating random variates from matrix-exponential distributions and phase-type distributions with non-Markovian initialisation vector is presented in [HT11]. Given a representation $(\boldsymbol{\alpha}, \mathbf{Q})$ for an ME distribution where $\mathbf{Q}$ is a sub-generator matrix, but $\boldsymbol{\alpha}$ may contain negative entries, the method is based on an expression of the density as a combination of PH densities $g_i(t)$,

$$\begin{aligned} f(t) &= \boldsymbol{\alpha}\, \mathrm{e}^{\mathbf{Q}t}(-\mathbf{Q}\mathbb{1}) && (3.8)\\ &= \sum_{i=1}^{n} \alpha_i g_i(t), && (3.9) \end{aligned}$$

where $g_i(t)$ is the density of the PH distribution with representation $(\mathbf{e}_i, \mathbf{Q})$:

$$g_i(t) = \mathbf{e}_i\, \mathrm{e}^{\mathbf{Q}t}(-\mathbf{Q}\mathbb{1}). \quad (3.10)$$

The sum in (3.9) is split into a part with positive coefficients α_i, identified by the index set $\mathcal{A}_+$ and a part with negative coefficients, with index set $\mathcal{A}_-$:

$$\begin{aligned} f(t) &= \sum_{i\in\mathcal{A}_+} \alpha_i g_i(t) + \sum_{i\in\mathcal{A}_-} \alpha_i g_i(t) && (3.11)\\ &= f_+(t) + f_-(t), && (3.12) \end{aligned}$$

where $f_+(t) = \sum_{i\in\mathcal{A}_+} \alpha_i g_i(t)$ is strictly positive and can be normalised to mass one, such that $\hat{f}(t)$ is a proper density:

$$\hat{f}(t) = \frac{1}{\sum_{i\in\mathcal{A}_+} \alpha_i} f_+(t). \quad (3.13)$$

The method proceeds by drawing a random variate x from $\hat{f}(t)$ (using any algorithm for generating PH-distributed random variates). x is accepted with probability

$$p = 1 + \frac{f_-(x)}{f_+(x)}, \quad (3.14)$$

and rejected with probability (note that $\frac{f_-(x)}{f_+(x)} \leq 0$):

$$1 - p = -\frac{f_-(x)}{f_+(x)}. \quad (3.15)$$

In abbreviated form, the algorithm is given by:

Procedure `AcceptanceRejection`$(\boldsymbol{\alpha}, \mathbf{Q})$:
while true **do**
  Draw an $\hat{f}(t)$-distributed sample x.
  $p := 1 - \frac{f_-(t)}{f_+(t)}$
  Draw a sample u from the uniform distribution on $(0, 1)$.
  if $u < p$ **then**
    break
  end if
end while
return x

The full version of the algorithm, as presented in [HT11], avoids drawing the random number u if $\mathcal{A}_- = \emptyset$ (i.e. if the acceptance probability is 1) and additionally checks if $f(t)$ is a proper density; this part is omitted here for brevity.

The method presented by Horváth and Telek offers two advantages. First, it enables drawing random variates from PH distributions with non-Markovian representation. As demonstrated in [HT11], this enables the following approach: The PH distribution is transformed to a structure that allows efficient random-variate generation, but may not have a Markovian initialisation vector. From this representation, random variates can then be drawn using the presented algorithm. Horváth and Telek discuss the use of both branch structures (using the extended Hyper-Feedback-Erlang representation, where several identical FE blocks are allowed per branch) and chain structures (using the FE-diagonal representation).

A second advantage of this method lies in its applicability to a wider class of distributions than those considered here, because the algorithm is also suited for drawing random variates from matrix-exponential (ME) distributions. By reducing the problem of drawing from ME distributions to that of drawing PH-distributed random variates, Horváth and Telek make it possible to apply the efficient methods available for the PH class to the ME class.

Costs of Acceptance/Rejection. Our discussion of the costs follows that given in [HT11]. The costs for the `AcceptanceRejection` method depend on the method that is used in the algorithm and on the representation. With respect to the acceptance probability p, we must distinguish two cases: If the representation is Markovian, then p will be equal to 1, in wich case the method reduces to a single call to the algorithm for drawing PH-distributed random variates. If the representation is not Markovian, then $p < 1$, and the PH sampling algorithm will be called multiple times.

If an FE-diagonal representation is used, the acceptance probability $p = 1$, since every PH distribution has a Markovian FE-diagonal representation (Section 2.4.2).

In this case, the method reduces to the `FE-diagonal` algorithm, described in Section 3.5.1.3. Both worst-case costs and average costs are then equal to those of the `FE-diagonal` method, which are given in Section 3.5.1.3.

If the extended Hyper-FE representation is used, it is possible that the representation is not Markovian. In this case, the acceptance probability p is less than 1, i.e. several random variates may be required before the sample is accepted. The costs for each trial are equal to the costs of the `eHFED` method (see Section 3.5.2.2). Furthermore, the number of trials that are required follows a geometric distribution with parameter p and support $1, 2, \ldots$. Since the geometric distribution has infinite support, we cannot define the maximum number of trials, and therefore the worst-case costs for this case are not defined. Average costs, on the other hand, can be derived using the mean $1/p$ of the geometric distribution: On average,

$$\#uni_{\text{Avg, AR/eHFED}} = \frac{1}{p}\#uni_{\text{Avg, eHFED}} \tag{3.16}$$

uniform random numbers are required and

$$\#ln_{\text{Avg, AR/eHFED}} = \frac{1}{p}\#ln_{\text{Avg, eHFED}} \tag{3.17}$$

logarithms will be computed. Expressions for the eHFED-method are given in Section 3.5.2.2.

3.5 Characterisation Methods for Random-Variate Generation

According to the classification in [Jai91], characterisation methods for random-variate generation are based on expressing the desired distribution using other distributions for which efficient methods are known. The definition of phase-type distributions as the distribution of the time to absorption in a Markov chain [Neu81] provides an intuitive characterisation of these distributions for the purposes of random-variate generation: Each sample is generated by entering the Markov chain at some state and traversing the chain, spending an exponentially-distributed time in each state, until the absorbing state is reached. The sample is then the sum of the exponential samples along the path to absorption. The methods discussed in this section are also referred to as 'Play' methods, since the above approach is equivalent to 'playing' the Markov chain till absorption. As these methods operate on the Markov chain, they can only be applied if the distribution is given in a Markovian representation. In this section we will show that sub-classes and special representations enable more efficient random-variate generation. Note that these algorithms may also be used as part of the acceptance-rejection method by [HT11].

The costs of the characterisation methods described here often depend on the number of state transitions. We therefore define $\tilde{n}$ to denote the maximum number of state transitions and n^* to denote the average.

3.5.1 Algorithms for General Phase-type Distributions

We first describe three algorithms for generating random variates from general phase-type distributions. We start with two algorithms that work directly on the state-space representation and can thus be used with any phase-type distribution. We then propose an algorithm that requires FE-diagonal representations and utilises their special structure to simplify the process of random-variate generation.

3.5.1.1 The Generic Play Method

Recall from Chapter 2 that phase-type distributions are defined as the distribution of time-to-absorption in a continuous-time Markov chain [Neu81]. Therefore, a sample from a PH distribution is given by the time spent between entering the chain at some state i and reaching the absorbing state $n+1$. The most natural way to generate a PH-distributed sample is then to 'play' the CTMC (hence the name of the method) by entering the chain at one state, following the state-transitions and summing up the state-sojourn times, until the absorbing state is reached. The PH sample is then the sum of the state-sojourn times. It appears that this method has been first mentioned in [NP81], although it was probably known before. The algorithm can be stated as follows:

Procedure `Play`$(\boldsymbol{\alpha}, \mathbf{Q})$:
$x := 0$
Draw an $\boldsymbol{\alpha}$-distributed discrete sample i for the initial state.
while $i \neq n+1$ **do**
  $x := x + \mathrm{Exp}(\lambda_{ii})$
  Draw a $\overline{\mathbf{P}}^{(i)}$-distributed discrete sample i for the next state.
end while
return x

In the `Play` algorithm, first an initial state i is selected randomly, with distribution according to the initial probability vector $\boldsymbol{\alpha}$. This is achieved by drawing an integer sample from the distribution in $\boldsymbol{\alpha}$. The probabilities of state transitions within the Markov chain are given by the entries of the transition matrix $\mathbf{P}$ of the embedded Markov chain (EMC) of the generator matrix $\hat{\mathbf{Q}}$ (see Section B.2 for the derivation of the EMC),

$$\mathbf{P} = \mathbf{I} - \mathrm{Diag}(\hat{\mathbf{Q}})^{-1}\hat{\mathbf{Q}},$$

where $\mathrm{Diag}(\hat{\mathbf{Q}})$ is the matrix with the diagonal elements of $\hat{\mathbf{Q}}$ on the diagonal and zero everywhere else. Let $\overline{\mathbf{P}}^{(i)}$ denote the ith row vector of $\mathbf{P}$. Then, if the chain

is in state i, the probability that $j \neq i$ is the next state is given by the jth entry of $\overline{\mathbf{P}}^{(i)}$. Thus, in order to select the next state, the procedure draws an integer sample with distribution $\overline{\mathbf{P}}^{(i)}$ (note that $\overline{\mathbf{P}}^{(i)}_i = 0$, i.e. the chain cannot stay in the current state). The sojourn times for each state i follow an exponential distribution with rate λ_{ii}. Therefore, in each state i the procedure draws a sample from the exponential distribution with rate λ_{ii}. When the absorbing state is reached, the procedure terminates and returns the sum of the exponential samples.

Costs of the generic Play method. Each traversed state requires one exponentially-distributed random variate, and hence in each step one uniform and one logarithm are computed. Additionally, one uniform is required for the initial state selection, and each successive state selection also requires a uniform random number. For this method, $\#uni$ and $\#ln$ are proportional to $\tilde{n}$. However, $\tilde{n}$ is not defined if there are cycles in the CTMC, since an arbitrary number of loops could be performed before absorption. Therefore, worst-case costs are not defined for `Play`.

The average number of state traversals, on the other hand, can be computed by scaling the generator matrix such that the mean sojourn time in each state is 1, and computing the mean of the resulting distribution:

$$n^* = \boldsymbol{\alpha}(\mathrm{Diag}(\mathbf{Q})^{-1}\mathbf{Q})^{-1}\mathbb{1}. \tag{3.18}$$

Since each step requires two random numbers and one logarithm, and an additional random number is drawn for selection of the first state, average costs are

$$\#uni_{\text{Avg, Play}} = 2n^* + 1 \tag{3.19}$$

$$\#ln_{\text{Avg, Play}} = n^*. \tag{3.20}$$

3.5.1.2 The Count Method

In [NP81], Neuts and Pagano observe that, when traversing a state more than once, the `Play` method adds up multiple samples from the same exponential distribution, computing one logarithm for each exponential random variate. They point out that the sum of k_i exponential distributions of the same rate λ_{ii} is the Erlang distribution with length k_i and rate λ_{ii}. As discussed above, drawing a sample from the Erlang distribution of length k_i requires only one logarithm operation, as opposed to k_i logarithms when drawing k_i individual exponential samples. In order to reduce the number of logarithms needed, Neuts and Pagano propose the following algorithm, which we will refer to as the `Count` method, because it counts the number of traversals for each state:

Procedure `Count`$(\boldsymbol{\alpha}, \mathbf{Q})$:
for $i = 1, \ldots, n$ **do**
  $k_i := 0$
end for
Draw an $\boldsymbol{\alpha}$-distributed discrete sample i for the initial state.
while $i \neq n + 1$ **do**
  $k_i := k_i + 1$
  Draw a $\overline{\mathbf{P}}^{(i)}$-distributed discrete sample i for the next state.
end while
$x := 0$
for $i = 1, \ldots, n$ **do**
  $x := x + \mathrm{Erl}(\lambda_{ii}, k_i)$
end for
return x

The `Count` method is equivalent to the `Play` method in that it plays the Markov chain by following state transitions as given by the embedded Markov chain. However, instead of drawing a sojourn time for each state i, the method only counts the number of visits to that state in a variable k_i. After the chain has reached the absorbing state, `Count` draws one $\mathrm{Erl}(\lambda_{ii}, k_i)$-distributed sample for each state $i = 1, \ldots, n$, and returns the sum of these samples.

Costs of the Count method. As with the `Play` method, the maximum number of state traversals cannot be specified, and thus the worst-case number of uniforms is not defined. On the other hand, `Count` requires only n Erlang-distributed samples (one for each state). This allows us to bound the maximum number of logarithm operations by the size of the representation (i.e. the number of states):

$$\#ln_{\text{WC, Count}} \quad = \quad n. \tag{3.21}$$

Average costs for the `Count` methods are as follows: Since states are traversed as in the `Play` method and one exponential sample is drawn for each state, the same number of uniforms is required:

$$\#uni_{\text{Avg, Count}} \quad = \quad 2n^* + 1. \tag{3.22}$$

The average number of logarithms, on the other hand, is equal to that of the worst case:

$$\#ln_{\text{Avg, Count}} \quad = \quad n. \tag{3.23}$$

3.5.1.3 The FE-Diagonal Method

Both `Play` and `Count` support general phase-type distributions in an arbitrary Markovian representation. In order to play the Markov chain, the methods have to draw one uniform random variate for selecting the next state in each step. We now describe a method that uses the FE-diagonal representation introduced in Section 2.3.2. Recall from Definition 2.3.5 that the FE-diagonal representation given by

$$(\boldsymbol{\alpha}, \boldsymbol{\Upsilon}) = ((\alpha_1, \ldots, \alpha_n), ((b_1, \lambda_1, z_1), \ldots, (b_m, \lambda_m, z_m)),$$

where b_i, λ_i and z_i denote the the length, rate, and feedback probability of the ith block, respectively, has chain structure, i.e. if it has been entered at some FE block $i, i = 1, \ldots, m$, all remaining FE blocks $j = i + 1, \ldots, m$ have to be traversed before the absorbing state is reached. Thus, the FE-diagonal structure eliminates the need for randomly selecting the next block. On the other hand, since every PH distribution can be transformed to the Monocyclic form, which is a special FE-diagonal representation (Section 2.4), an algorithm that employs the FE-diagonal form still supports the whole PH class.

The procedure shown in the following was first proposed by us in [RWBT09] for use with Monocyclic representations. Since the algorithm only depends on the FE-diagonal structure and not on the ordering of the eigenvalues, it is renamed to `FE-diagonal` here.

Procedure `FE-diagonal`$(\boldsymbol{\alpha}, \boldsymbol{\Upsilon})$:
$x := 0$
Draw an $\boldsymbol{\alpha}$-distributed discrete sample i for the initial state.
$j := \text{argmin}_j \left\{ j = 1, \ldots, m \mid i \leq \sum_{h=1}^{j} b_h \right\}$
$l := b_j - (i - i_j)$
while $j \leq m$ **do**
  $c := \text{Geo}(1 - z_j)$
  $x := x + \text{Erl}(\lambda_i, cb_j + l)$
  $j := j + 1$
  $l := b_j$
end while
return x

The algorithm works as follows: First, an initial state is chosen according to the initial probability vector $\boldsymbol{\alpha}$. This state belongs to FE block j, whose first state is denoted by i_j. There are $1 \leq l \leq b_j$ states to traverse before the chain could leave this block: If the block was entered at the last state, only this state needs to be traversed, i.e. $l = 1$; if the block was entered at its first state, i.e. $i = i_j$, then the whole block has to be traversed, and thus $l = b_j$ for this case. Since all rates in the block are equal (λ_j), this first traversal corresponds to an Erlang(λ_j, l) distribution.

When the last state of the block is reached, one may either enter the next block or follow the feedback-loop to the first state of the current block. The number of loops $c = 0, 1, \ldots$ within the jth block follows a geometric distribution with parameter $1 - z_j$. The random variate corresponding to the loop is Erlang-(λ_j, cb_j)-distributed. Consequently, for the block entered upon initialisation the algorithm draws a random variate from an Erlang-$(\lambda_j, cb_j + l)$ distribution. All the remaining blocks until absorption are entered at the first state, and thus the respective random variates for blocks $j' = j + 1, \ldots, m$ are distributed according to $(\mathbf{e}_1, \mathbf{F}_{j'})$ distributions, where $\mathbf{e}_1$ is the row vector with 1 at position 1 and zero everywhere else and $\mathbf{F}_{j'}$ is the sub-generator corresponding to the j'th FE block. Following the argument for the initial step, random variates from these distributions are generated from Erlang-$(\lambda_{j'}, c_{j'} b_{j'} + l_{j'})$ distributions, where $c_{j'}$ is the number of loops and $l_{j'} = b_{j'}$ (since each block is entered at the beginning and has to be traversed at least once). By summing up all these random variates, the final random variate is obtained.

Costs of the FE-diagonal method. Due to the possible presence of cycles in the FE-diagonal representation, the maximum number of state traversals is not defined, and thus we cannot define the worst-case for the number of uniforms. The maximal number of block traversals, on the other hand, occurs if the chain is entered at the first block. As each block requires one Erlang-distributed random variate and one Geometric sample, three logarithms are computed for each block. For m blocks, the worst-case costs for the number of logarithms are thus

$$\#ln_{\text{WC, FE-diagonal}} \quad = \quad 3m. \tag{3.24}$$

In order to derive the average costs, we first introduce the following vectors:

$$\bar{\boldsymbol{\alpha}} = \left(\boldsymbol{\alpha}^{(1)}\mathbb{1}, \ldots, \boldsymbol{\alpha}^{(m)}\mathbb{1}\right) \in \mathbb{R}^m, \tag{3.25}$$

$$\boldsymbol{\mu} = (b_1, b_1 - 1, \ldots, 1, b_2, \ldots, 1, b_m, \ldots, 1) \in \mathbb{R}^n, \tag{3.26}$$

$$\boldsymbol{\nu} = (m, m - 1, \ldots, 1) \in \mathbb{R}^m, \tag{3.27}$$

$$\boldsymbol{\nu}^{(1)} = \left(\frac{z_1}{1 - z_1} b_1, \ldots, \frac{z_m}{1 - z_m} b_m\right) \in \mathbb{R}^m, \tag{3.28}$$

$$\boldsymbol{\nu}^{(2)} = \left(\sum_{j=2}^{m} \frac{1}{1 - z_j} b_j, \ldots, \frac{1}{1 - z_{m-1}} b_{m-1}, 0\right) \in \mathbb{R}^m. \tag{3.29}$$

The entry $\bar{\alpha}_j$ of $\bar{\boldsymbol{\alpha}}$ denotes the initial probability of the jth Feedback-Erlang block. The vector $\boldsymbol{\mu}$ contains the number of phases that have to be traversed before the next feedback-point is reached, i.e. if the chain is entered at state i belonging to block j, μ_i gives the number of phases remaining until the end of the jth block. Entry $\boldsymbol{\nu}_j$ of $\boldsymbol{\nu}$ describes the number of *blocks* that have to be traversed if the chain is

entered at the jth block. The vectors $\boldsymbol{\nu}^{(1)}$ and $\boldsymbol{\nu}^{(2)}$ summarise the average number of state transitions within blocks. $\boldsymbol{\nu}^{(1)}$ gives the average number of phases to be traversed within each FE block due to loops, i.e. the average number of phases if the block is entered at its last state. $\boldsymbol{\nu}^{(2)}$, on the other hand, specifies the average number of state transitions in the remaining blocks $j+1,\ldots,m$ when the chain has been entered at block j.

Using these vectors, we obtain the following expression for the average number of state transitions with an FE-Diagonal representation:

$$n^*_{\text{FE-Diagonal}} \quad = \quad \boldsymbol{\alpha}\boldsymbol{\mu}^{\mathsf{T}} + \bar{\boldsymbol{\alpha}}\left(\boldsymbol{\nu}^{(1)} + \boldsymbol{\nu}^{(2)}\right)^{\mathsf{T}}. \tag{3.30}$$

The first term in (3.30) describes the average number of state transitions until a feedback loop is reached, i.e. with probability α_i $\boldsymbol{\mu}_i$ states have to be traversed. $\bar{\boldsymbol{\alpha}}(\boldsymbol{\nu}^{(1)})^{\mathsf{T}}$ gives the average number of states visited in the loop corresponding to the initial FE block, and $\bar{\boldsymbol{\alpha}}(\boldsymbol{\nu}^{(2)})^{\mathsf{T}}$ is the average number of states in the remaining FE blocks.

The average number of FE blocks that are visited is determined by the initial block probabilities. Due to the chain structure of the FE diagonal representation, whenever a block j is entered, all blocks $j, j+1,\ldots,m$ have to be traversed. Thus the average number of FE blocks is

$$\ell^* \quad = \quad \bar{\boldsymbol{\alpha}}\boldsymbol{\nu}^{\mathsf{T}}. \tag{3.31}$$

As the `FE-Diagonal` method requires one uniform random number for the initial block selection, one uniform for drawing the length of the Erlang in each traversed block, and one uniform random number for each visited state, we obtain

$$\#uni_{\text{Avg, FE-Diagonal}} \quad = \quad 1 + \ell^* + n^*_{\text{FE-Diagonal}}. \tag{3.32}$$

Furthermore, since each visited FE block requires three logarithm operations,

$$\#ln_{\text{Avg, FE-Diagonal}} \quad = \quad 3\ell^*. \tag{3.33}$$

Note that with slight modifications the `FE-Diagonal` algorithm could avoid drawing a geometric sample for degenerate Feedback-Erlang blocks (i.e. blocks with feedback probability $z = 0$). An algorithm with these modifications would result in a different expression for the average number of logarithms, as given in [HRTW12]. However, as in the case of partial pre-computation for geometric random variate generation, we omit such optimisations here in order to obtain simple expressions.

3.5.2 Algorithms for Sub-Classes

The three methods described in the previous section are applicable to general PH distributions. We now present methods that exploit special structures available for the Acyclic Phase-type (APH), Hyper-Erlang (HErD), and Hyper-Feedback-Erlang (HFED) distributions. These sub-classes are well-supported by fitting-tools (see Section 5.2), and their low number of parameters and special structure help to reduce the number of atomic operations that are required.

3.5.2.1 The SimplePlay Method

We first consider the APH class. Recall from Section 2.4 that every APH distribution has a representation in CF-1 form, which is a special case of the bi-diagonal representations discussed in Section 2.3.2, where we introduced the notation (Definition 2.3.4)

$$(\boldsymbol{\alpha}, \boldsymbol{\Lambda}) = ((\alpha_1, \ldots, \alpha_n), (\lambda_1, \ldots, \lambda_n)).$$

Bi-diagonal representations are a special class of FE-diagonal representations where all FE blocks are of size 1, i.e. $m = n$. Following the argument we employed in the discussion of the `FE-diagonal` method, no random numbers are required for determining the next state when playing the Markov chain, since each state has exactly one successor state. Furthermore, in a bi-diagonal representation all FE blocks have length 1 and feedback probability 0. Thus the state sojourn time for each block j simply follows an exponential distribution with rate λ_j. Consequently, once an initial state has been selected, the random variate is simply the sum of exponentially distributed samples from each of the successor states. In pseudo-code, the `SimplePlay` method is described as follows [RWBT09]:

Procedure `SimplePlay`$(\boldsymbol{\alpha}, \boldsymbol{\Lambda})$:
$x := 0$
Draw an $\boldsymbol{\alpha}$-distributed discrete sample i_S for the initial state.
for $i = i_S, \ldots, n$ **do**
  $x := x + \mathrm{Exp}(\lambda_i)$
end for
return x

The procedure first selects an initial state. It then draws $\mathrm{Exp}(\lambda_i)$-distributed sojourn times for each of the $n - i + 1$ states traversed until absorption and returns the result. Note that the transition rates are not guaranteed to be identical, and hence simplification to drawing an Erlang sample is, in general, not possible.

Costs of the SimplePlay method. In contrast to the representations discussed before, the bi-diagonal structures cannot contain cycles. Therefore, the maximum number of state transitions is defined. The worst case occurs if the chain is entered

at the first state and all states have to be traversed. Therefore, the worst-case number of state transitions is $\tilde{n}_{\text{WC, SimplePlay}} = n$, and the worst-case costs for the number of uniforms and the number of logarithms are given by

$$\#uni_{\text{WC, SimplePlay}} = \tilde{n}_{\text{WC, SimplePlay}} + 1 = n + 1 \text{ and} \tag{3.34}$$

$$\#ln_{\text{WC, SimplePlay}} = \tilde{n}_{\text{WC, SimplePlay}} = n. \tag{3.35}$$

The bi-diagonal form is a special case of the FE-diagonal form, and thus for the average costs we can apply the same argument as the one we used for deriving the costs of the `FE-diagonal` algorithm. Since the bi-diagonal form has $m = n$, $b_i = 1$ and $z_i = 0$ for $i = 1, \ldots, m$, (3.30) reduces to

$$n^*_{\text{SimplePlay}} = \boldsymbol{\alpha}\boldsymbol{\mu}^{\mathsf{T}} + \boldsymbol{\alpha}(\boldsymbol{\nu}^{(2)})^{\mathsf{T}} = \boldsymbol{\alpha}\boldsymbol{\nu}^{\mathsf{T}}. \tag{3.36}$$

Since `SimpleCount` does not draw geometric random variates and thus only requires one uniform random number and one logarithm for each traversed step, we obtain

$$\#uni_{\text{Avg, SimplePlay}} = n^*_{\text{SimpleCount}} + 1 = \boldsymbol{\alpha}\boldsymbol{\nu}^{\mathsf{T}} + 1, \tag{3.37}$$

$$\#ln_{\text{Avg, SimplePlay}} = n^*_{\text{SimpleCount}} = \boldsymbol{\alpha}\boldsymbol{\nu}^{\mathsf{T}}, \tag{3.38}$$

that is, apart from the constant offset 1 implied by the additional uniform random number required for initial state selection, both metrics are equal for bi-diagonal representations.

3.5.2.2 The HFED and eHFED Methods

Hyper-Feedback-Erlang representations, as defined in Definition 2.3.7, consist of parallel Feedback-Erlang blocks in branch structure. Drawing a random variate requires selecting a branch and then drawing a random variate from that branch. Drawing a random variate from a branch involves the same operation as for the first FE block in `FE-diagonal`. The algorithm is shown in the following:

Procedure `HFED`$(\boldsymbol{\alpha}, \boldsymbol{\Lambda})$:
Draw an $\boldsymbol{\alpha}$-distributed discrete sample i for the initial state.
$k := \text{argmin}_k \left\{ k = 1, \ldots, m \mid \sum_{h=1}^{k} b_k \right\}$
$l := b_k - (i - i_k)$
$c := \text{Geo}(1 - z_k)$
$x := \text{Erl}(\lambda_k, cb_k + l)$
return x

Horváth and Telek give a more general version of this algorithm in [HT11] for the generation of samples from extended Hyper-Feedback-Erlang representations:

Procedure `eHFED`$(\boldsymbol{\alpha}, \boldsymbol{\Upsilon})$:
$x := 0$
Draw an $\boldsymbol{\alpha}$-distributed discrete sample for the initial state.
$k := \text{argmin}_k \left\{ k = 1, \ldots, m \mid i \leq \sum_{h=1}^{k} b_h \right\}$
$j := \text{argmin}_j \{ j = 1, \ldots, m_k \mid i \leq i_k + j b_k \}$
$l := b_k - (i - i_{k,j})$
$h := m_k - j + 1$
$c := l + (h-1)b_k + \sum_{o=1}^{h} \text{Geo}(1 - z_k) b_k$
$x := \text{Erl}(\lambda_b, c)$
return x

The eHFED algorithm can be derived as follows: First, note that each branch in an eHFED representation is in FE-diagonal form. Consequently, drawing a random variate requires choosing a branch k and then drawing a sample from the FE-diagonal representation corresponding to this branch. Since all FE blocks of a branch are identical, drawing an FE-diagonal sample simplifies to generating a single sample from an Erlang distribution. Assuming that the FE-diagonal branch k has been entered at FE block j and state $i = 1, \ldots, b_k$ of that block, the number of states to traverse until the first feedback is reached is

$$l \quad = \quad b_k - (i - i_{k,j}), \tag{3.39}$$

where

$$i_{k,j} \quad = \quad \sum_{h=1}^{k-1} b_k + (j-1)b_k + 1 \tag{3.40}$$

denotes the first state of the jth block in the kth branch. The number of blocks that have to be traversed before absorption (counting the initial block) is

$$h \quad = \quad m_k - j + 1. \tag{3.41}$$

As with the `FE-diagonal` algorithm, the number of loops in each block follows a geometric distribution on $0, 1, \ldots$ with parameter $1 - z_k$, and each block except the initial one has to be traversed fully at least once. Since all FE blocks in a branch have the same feedback probability z_k, the length of the Erlang distribution is

$$c \quad = \quad l + (h-1)b_k + \sum_{o=1}^{h} \text{Geo}(1 - z_k) b_k. \tag{3.42}$$

Costs of the HFED and eHFED methods. The worst-case number of state traversals for the `HFED` and `eHFED` methods cannot be defined, as the FE blocks may

contain loops. The worst-case number of block traversals, however, can be derived: For the `eHFED` method, the worst case occurs if the branch i with the largest number of blocks m_i is chosen (for the `HFED` method, all branches have the same number of blocks). As each block requires two logarithms for computation of the geometric random variate, and one logarithm is needed for drawing the Erlang random variate, worst-case logarithm costs are

$$\#ln_{\text{WC, HFED}} = 3 \tag{3.43}$$

$$\#ln_{\text{WC, eHFED}} = 1 + 2 \max_{i=1,\ldots,m} \{m_i\}. \tag{3.44}$$

In order to derive expressions for the average costs of the `HFED` algorithm, we apply the following argument: Drawing a HFED-distributed random variate using `HFED` is equivalent to selecting a state in a Feedback-Erlang block and drawing a random variate from the initial block in an FE-diagonal representation using `FE-Diagonal` and then immediately entering the absorbing state. Consequently, the number of state traversals is equal to the sum of the average number of state traversals required to reach the feedback point when starting from any state in the branch and the average number of visits incurred by the feedback loop:

$$n^*_{\text{HFED}} = \bar{\boldsymbol{\alpha}} \left(\boldsymbol{\mu} + \boldsymbol{\nu}^{(1)} \right)^{\mathsf{T}}, \tag{3.45}$$

while the number of visited blocks is constant:

$$\ell_{\text{HFED}} = 1. \tag{3.46}$$

Two uniform random numbers are required for the selection of the initial state and for drawing the number of loops in the block. As only a single block is traversed, the algorithm draws one geometric random variate and computes one logarithm for the Erlang random variate. Therefore, average costs for the `HFED` method are:

$$\#uni_{\text{Avg, HFED}} = n^*_{\text{HFED}} + 2, \tag{3.47}$$

$$\#ln_{\text{Avg, HFED}} = 3. \tag{3.48}$$

Expressions for the average costs of the `eHFED` algorithm are given in [HT11]. In the following we derive more compact expressions, but remark that these are equivalent to those in [HT11]. Since the `eHFED` algorithm uses FE-diagonal representations in the branches, we can employ 3.25–3.29 to obtain the average costs. Let $n_k = m_k b_k$ be the length of the kth branch. For the kth branch, we modify the definitions

of 3.25–3.29 as follows:

$$\bar{\boldsymbol{\alpha}}^{(k)} = \left(\hat{\boldsymbol{\alpha}}^{(k,1)}\mathbb{1}, \ldots, \hat{\boldsymbol{\alpha}}^{(k,m_k)}\mathbb{1}\right) \in \mathbb{R}_k^m, \tag{3.49}$$

$$\boldsymbol{\mu}^{(k)} = (b_k, b_k - 1, \ldots, 1, b_k, b_k - 1, \ldots, 1, b_k, \ldots, 1) \in \mathbb{R}^{n_k}, \tag{3.50}$$

$$\boldsymbol{\nu}^{(k)} = (m_k, m_k - 1, \ldots, 1) \in \mathbb{R}^{m_k}, \tag{3.51}$$

$$\boldsymbol{\nu}^{(1,k)} = \left(\frac{z_k}{1 - z_k} b_k, \ldots, \frac{z_k}{1 - z_k} b_k\right) \in \mathbb{R}^{m_k}, \tag{3.52}$$

$$\boldsymbol{\nu}^{(2,k)} = \left(\frac{m_k - 1}{1 - z_k} b_k, \frac{m_k - 2}{1 - z_k} b_k, \ldots, \frac{1}{1 - z_k} b_k, 0\right) \in \mathbb{R}^{m_k}. \tag{3.53}$$

The vector $\boldsymbol{\alpha}^{(k)}$ summarises the initial probabilities for each of the FE blocks of the kth branch. The ith entry $\boldsymbol{\mu}^{(k)}$ gives the number of states to be traversed until the next feedback-point can be reached if the kth branch is entered at the ith state. The repetitive structure of this vector is due to the fact that all FE blocks in the kth branch are identical. The jth entry of $\boldsymbol{\nu}^{(k)}$ describes the number of FE blocks left before absorption if the kth branch is entered at FE block j. $\boldsymbol{\nu}^{(1,k)}$ gives the expected number state traversals due to taking the feedback loop for each block. Again, the repetitive structure reflects that the FE blocks are identical. Finally, the jth entry of $\boldsymbol{\nu}^{(2,k)}$ defines the average number of state traversals in the blocks $j + 1, j + 2, \ldots, m_k$ which are traversed fully if the kth branch was entered at the jth block.

Based on these definitions, we can express the average number of state traversals in the kth branch as

$$n_k^* = \hat{\boldsymbol{\alpha}}^{(k)} \left(\boldsymbol{\mu}^{(k)}\right)^{\mathsf{T}} + \bar{\boldsymbol{\alpha}}^{(k)} \left(\boldsymbol{\nu}^{(1,k)} + \boldsymbol{\nu}^{(2,k)}\right). \tag{3.54}$$

The average number of state traversals is the weighted sum of the state traversals for each branch:

$$n_{\text{eHFED}}^* = \sum_{k=1}^{m} \bar{\boldsymbol{\alpha}}^{(k)} \mathbb{1} n_k^*. \tag{3.55}$$

Likewise,

$$\ell_k^* = \bar{\boldsymbol{\alpha}}^{(k)} (\boldsymbol{\mu}^{(k)})^{\mathsf{T}} \tag{3.56}$$

gives the expected number of block traversals in the kth branch, and

$$\ell_{\text{eHFED}}^* = \sum_{k=1}^{m} \bar{\boldsymbol{\alpha}}^{(k)} \mathbb{1} \ell_k^* \tag{3.57}$$

is the overall average number of block traversals.

For the average number of random uniform numbers we then obtain

$$\#uni_{\text{Avg,eHFED}} = 1 + \ell^*_{\text{eHFED}} + n^*_{\text{eHFED}} \tag{3.58}$$

while the number of logarithms is

$$\#ln_{\text{Avg,eHFED}} = 1 + 2\ell^*_{\text{eHFED}}, \tag{3.59}$$

since each Geometric random variate requires two logarithms, and one logarithm is needed for the single Erlang sample.

3.5.2.3 The SimpleCount Method

We conclude this section by describing an algorithm for drawing random variates from a Hyper-Erlang distribution. Hyper-Erlang distributions can be represented in branch structure, where every branch $k \in \{1, \ldots, m\}$ is an Erlang distribution with length b_k, rate λ_k, and the branch probability given by the entries β_k of the vector $\boldsymbol{\beta}$. In Chapter 2 we denoted this representation by

$$(\boldsymbol{\beta}, \mathbf{E}) = ((\beta_1, \ldots, \beta_m), ((b_1, \lambda_1), \ldots, (b_m, \lambda_m))\,.$$

Generating a random variate from this representation is straightforward: After selecting a branch $k \in \{1, \ldots, m\}$ by drawing a sample from the discrete distribution defined by the probability vector $\boldsymbol{\beta}$, an $\text{Erl}(b_k, \lambda_k)$-distributed sample is drawn and returned [RWBT09]:

Procedure `SimpleCount`$(\boldsymbol{\beta}, \mathbf{E})$:
Draw a $\boldsymbol{\beta}$-distributed discrete sample k
return $\text{Erl}(\lambda_k, b_k)$

Costs of the SimpleCount method. Hyper-Erlang distributions do not contain cycles, and therefore worst-case costs can be easily computed: The worst case occurs if the longest branch is entered, and therefore $\tilde{n}_{\text{WC, SimpleCount}} = \max_{i=k,\ldots,m}\{b_k\}$. Since each branch requires exactly one logarithm, worst-case costs are as follows:

$$\#uni_{\text{WC, SimpleCount}} = 1 + \tilde{n}_{\text{WC, SimpleCount}} \text{ and} \tag{3.60}$$

$$\#ln_{\text{WC, SimpleCount}} = 1. \tag{3.61}$$

Average costs are also straightforward: Because a Hyper-Erlang distribution is a special Hyper-Feedback-Erlang distribution with all feedback probabilities equal to zero, and since blocks can only be entered at the first state, (3.45) simplifies to

$$n^*_{\text{SimpleCount}} = \bar{\boldsymbol{\alpha}}\boldsymbol{\mu} = \boldsymbol{\beta}\mathbf{b}^{\mathsf{T}}, \tag{3.62}$$

and we obtain

$$\#uni_{\text{Avg, SimpleCount}} = \boldsymbol{\beta}\boldsymbol{b}^{\mathsf{T}} + 1, \tag{3.63}$$

$$\#ln_{\text{Avg, SimpleCount}} = 1. \tag{3.64}$$

3.6 Example: Costs of Random-Variate Generation

In the previous sections we surveyed methods for random-variate generation and proposed the `FE-diagonal` and `SimplePlay` methods, which use the FE-diagonal and bi-diagonal forms, respectively. We derived expressions for the costs of the algorithms, which vary between methods and depend on the representations. We will now shed more light on the influences of representations and methods, by discussing the costs for a concrete example. Consider the distribution with representation $(\boldsymbol{\alpha}, \mathbf{Q})$ where

$$\boldsymbol{\alpha} = (0.1, 0, 0.9, 0, 0, 0) \quad \text{and} \quad \mathbf{Q} = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 \\ 0 & 0 & 0 & -2 & 2 \\ 0 & 0 & 0 & 0 & -2 \end{pmatrix}. \tag{3.65}$$

$(\boldsymbol{\alpha}, \mathbf{Q})$ is a Hyper-Erlang distribution of size $n = 5$ with two branches of length $b_1 = 2, b_2 = 3$, given in Hyper-Erlang form. We choose a Hyper-Erlang distribution because it is supported by all of the algorithms we studied in this chapter. Usually, we would employ the `SimpleCount` method to generate random variates from this distribution. However, in order to illustrate the impact of the representations and the methods on the cost of random-variate generation, in the following we assume that we do not know the structure of the distribution and apply the methods as in the most general case they support.

Some of the methods require special representations; in particular, `SimpleCount`, `HFED` and `eHFED` need the Erlang, Hyper-FE, and extended Hyper-FE forms, respectively, and `SimplePlay` and `FE-diagonal` require the bi-diagonal and FE-diagonal representations. Since $(\boldsymbol{\alpha}, \mathbf{Q})$ is a Hyper-Erlang distribution in Hyper-Erlang form, $(\boldsymbol{\alpha}, \mathbf{Q})$ is also in HFED and eHFED form. Furthermore, $(\boldsymbol{\alpha}, \mathbf{Q})$ is an acyclic phase-type distribution, and consequently it has a (bi-diagonal) CF-1 representation, which is identical to the (FE-diagonal) Monocyclic representation for this distribution. The

CF-1 form is

$$\boldsymbol{\alpha}' = (0.0125, 0.0375, 0.925, 0.025, 0) \quad \text{and} \quad \mathbf{Q}' = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 \\ 0 & 0 & 0 & -2 & 2 \\ 0 & 0 & 0 & 0 & -2 \end{pmatrix}. \tag{3.66}$$

The initial vector $\boldsymbol{\alpha}' := \boldsymbol{\alpha}\mathbf{S}$ is obtained using the similarity transformation matrix $\mathbf{S}$ that transforms $\mathbf{Q}$ to $\mathbf{Q}'$ (see Section 2.3.1). For methods that support both representations we will present the costs for both.

3.6.1 Worst-Case Costs

We start with the worst-case costs. Using the `NumericalInversion` method, we need one uniform random number and $19n$ evaluations of the matrix exponential. Assuming that the matrix exponential is computed using scalar exponentials, we need n evaluations of the exponential function, i.e. $\#exp_{\text{WC, NumericalInversion}} = 19 \cdot 5 = 95$. This cost does not depend on the structure of the representation, because computation of the matrix exponential does not take into account the structure of the matrix.

For the generic `Play` method, worst-case costs are not defined in general. In our example, however, we can compute them, as neither of the representations has cycles. For $(\boldsymbol{\alpha}, \mathbf{Q})$ the worst case occurs when the second branch is entered. The method then draws one uniform random number to select the branch, one uniform random number for the sojourn times in each state, and one uniform to select the successor state. It thus requires $\#uni_{\text{WC, Play}} = 1 + 6$ uniforms. Furthermore, because in each state an exponentially-distributed random sample is drawn, $\#ln_{\text{WC, Play}} = 3$ logarithms need to be computed. If we instead use the $(\boldsymbol{\alpha}', \mathbf{Q}')$ representation, the worst case happens when the CTMC is entered at the first state, because then all the remaining states have to be traversed. Worst-case costs are then $\#uni'_{\text{WC, Play}} = 1 + 10$ and $\#ln'_{\text{WC, Play}} = 5$.

The `Count` method traverses the Markov chain in the same way as `Play`, but draws exactly one logarithm for each state of the chain, irrespective of how often that state was visited. Therefore, worst-case uniform costs for `Count` in our example are the same as for `Play`, while the number of logarithms $\#ln_{\text{WC, Count}} = 5$ for both representations. Note that for the $(\boldsymbol{\alpha}, \mathbf{Q})$ representation `Count` actually computes more logarithms than `Play`.

The `FE-diagonal` method only supports FE-diagonal representations. As with the `Play` and `Count` methods, worst-case costs for the number of uniforms are in general not defined for `FE-diagonal`, but can be computed for $(\boldsymbol{\alpha}', \mathbf{Q}')$, which contains

no cycles. As before, the worst case occurs if the CTMC is entered at the first state (i.e. the first FE-block). Then, one uniform is required for the choice of the initial state, one uniform is required for each traversed state, and one uniform is needed for drawing the length of the Erlang in each block (we assume that no optimisation for zero feedback probability takes place). Note that no uniforms are drawn for selection of the next state, since for each state its successor is encoded in the matrix $\mathbf{Q}'$. Consequently, the worst-case number of uniforms is $\#uni_{\text{WC, FE-diagonal}} = 1 + 5 + 2 = 8$. Recall that with `FE-diagonal` three logarithms are computed for each Feedback-Erlang block (no pre-computation of logarithms is applied). Worst-case logarithm costs are thus $\#ln_{\text{WC, FE-diagonal}} = 2 \cdot 3$.

Using the `SimplePlay` method on $(\boldsymbol{\alpha}', \mathbf{Q}')$, the longest path through the CTMC is again taken if the chain is entered at the first block. As `SimplePlay` does not consider FE blocks, no geometric random variate must be computed. We thus require one uniform for the initial state and one uniform for each state traversal, and therefore $\#uni_{\text{WC, SimplePlay}} = 1 + 5 = 6$. Because one logarithm is computed for each state, the worst-case number of logarithms is $\#ln_{\text{WC, SimplePlay}} = 5$.

The `HFED` and `eHFED` methods can only be used with (extended) Hyper-Feedback Erlang distributions. In our example, $(\boldsymbol{\alpha}, \mathbf{Q})$ is an eHFED distribution with block lengths $b_1 = 2, b_2 = 3$ and block multiplicity 1 in each branch; consequently, both algorithms have the same costs. Furthermore, since the feedback probabilities are zero in both blocks we can compute worst-case uniform costs, even though this is not possible in the general case. The worst case is equivalent to entering the longest branch. In this case, one uniform is required for state selection, three uniforms are needed for the traversed states, and one uniform is drawn for the geometric random variate. Worst-case uniform costs are thus $\#uni_{\text{WC, (e)HFED}} = 1 + 3 + 1 = 5$. As each branch implies three logarithms, and exactly one branch is entered, $\#ln_{\text{WC, (e)HFED}} = 3$.

With `SimpleCount`, in the worst case we draw one uniform random number for selection of the branch and three uniform random numbers for the states of the branch, resulting in $\#uni_{\text{WC, SimpleCount}} = 4$. As only one logarithm is computed for the whole branch, $\#ln_{\text{WC, SimpleCount}} = 1$.

Finally, we note that with either of the methods described here the acceptance probability in the `AcceptanceRejection` algorithm is equal to one, since all of the required representations are Markovian. Costs for the `AcceptanceRejection` method are then equal to those of the respective method. The abbreviated version of the method presented here requires an additional uniform for deciding whether the sample is accepted.

3.6.2 Average Costs

Average costs for `NumericalInversion` are equal to the worst-case costs for both representations: $\#uni_{\text{WC, NumericalInversion}} = 1$ and $\#exp_{\text{WC, NumericalInversion}} = 95$.

For the `Play` method, the average costs depend on the average number of state traversals, n^*. With $(\boldsymbol{\alpha}, \mathbf{Q})$, $n^* = 2.9$, while with $(\boldsymbol{\alpha}', \mathbf{Q}')$, $n^{*\prime} = 3.0375$. As each state requires two uniforms (one for the exponential sample, one for the selection of the next state), and an additional uniform is needed for initial state selection, $(\boldsymbol{\alpha}, \mathbf{Q})$ needs $\#uni_{\text{Avg, Play}} = 1 + 2n^* = 6.8$ uniform random numbers, while $(\boldsymbol{\alpha}', \mathbf{Q}')$ needs $\#uni'_{\text{Avg, Play}} = 7.075$. Likewise, the logarithm costs are $\#ln_{\text{Avg, Play}} = n^* = 2.9$ and $ln'_{\text{Avg, Play}} = 3.0375$, respectively.

With the `Count` algorithm, the average number of uniforms is equal to that of the `Play` algorithm. On the other hand, one logarithm is computed for each state, and therefore $\#ln_{\text{Avg, Play}} = \#ln'_{\text{Avg, Play}} = n = 5$.

`FE-diagonal` can only be applied to the $(\boldsymbol{\alpha}', \mathbf{Q}')$ representation. As this is a chain structure without any feedbacks, $n^{*\prime} = \boldsymbol{\alpha}(5, 4, 3, 2, 1)^\mathsf{T} = 3.0375$. `FE-diagonal` draws one uniform random number for the initial state, one uniform for each traversed state, and one uniform for the geometric random variate in each block. In our example, the average number of traversed blocks is $\ell^{*\prime} = (\alpha_1 + \alpha_2, \alpha_3 + \alpha_4 + \alpha_5)(2, 1)^\mathsf{T} = 1.05$. Consequently, $\#uni'_{\text{Avg, FE-diagonal}} = 1 + n^{*\prime} + \ell^{*\prime} = 5.0875$. Since each block implies the computation of three logarithms, $\#ln'_{\text{Avg, FE-diagonal}} = 3\ell^{*\prime} = 3.15$.

With the `SimplePlay` method, we traverse the same number of states as with `FE-diagonal`, but only need one uniform random number for each state and no uniforms for geometric samples. Therefore, $\#uni_{\text{Avg, SimplePlay}} = 1 + n^{*\prime} = 4.0375$. As we need one logarithm for each state, $\#ln'_{\text{Avg, SimplePlay}} = n^{*\prime} = 3.0375$.

As in the worst case, the `HFED` and `eHFED` methods have equal average costs for $(\boldsymbol{\alpha}, \mathbf{Q})$. Average costs for the uniform random numbers depend on the average number of states, furthermore, two additional uniforms are required (one for initial state selection, one for the geometric random variate). On average, $(\alpha_1+\alpha_2, \alpha_3+\alpha_4+\alpha_5)(2, 3)^\mathsf{T} = 2.9$ states are traversed; consequently, $\#uni_{\text{Avg, (e)HFED}} = 2.9+2 = 4.9$. The average number of logarithms is $\#ln_{\text{Avg, (e)HFED}} = 3$ (two for the geometric random variate, one for the Erlang sample).

If we use the `SimpleCount` method instead, we do not need the random uniforms and the logarithms for the geometric random variates. The cost metrics are therefore $\#uni_{\text{Avg, SimpleCount}} = 1 + 2.9 = 3.9$ and $\#ln_{\text{Avg, SimpleCount}} = 1$.

For `Acceptance/Rejection`, we have the same observation as in the worst case: The costs are again equal to the costs of the method used, and the abbreviated version of the algorithm shown here uses one additional uniform random number.

3.6.3 Discussion

We summarise the costs in Table 3.1. As remarked in our discussion, costs for `AcceptanceRejection` are equal to the costs of the algorithm used within the

Method	Worst Case				Average Case			
	$(\boldsymbol{\alpha}, \mathbf{Q})$		$(\boldsymbol{\alpha}', \mathbf{Q}')$		$(\boldsymbol{\alpha}, \mathbf{Q})$		$(\boldsymbol{\alpha}', \mathbf{Q}')$	
	$\#uni$	$\#exp$	$\#uni$	$\#exp$	$\#uni$	$\#exp$	$\#uni$	$\#exp$
`Numerical-Inversion`	1	95	1	95	1	95	1	95
	$\#uni$	$\#ln$	$\#uni$	$\#ln$	$\#uni$	$\#ln$	$\#uni$	$\#ln$
`Play`	7	3	11	5	6.8	2.9	7.075	3.0375
`Count`	7	5	11	5	6.8	5	7.075	5
`FE-diagonal`	–	–	8	6	–	–	5.0875	3.15
`SimplePlay`	–	–	6	5	–	–	4.0375	3.0375
`(e)HFED`	5	3	–	–	4.9	3	–	–
`SimpleCount`	4	1	–	–	3.9	1	–	–

Table 3.1: Example costs for a Hyper-Erlang distribution. $(\boldsymbol{\alpha}, \mathbf{Q})$ is in Hyper-Erlang form, while $(\boldsymbol{\alpha}', \mathbf{Q}')$ is in CF-1 form. The `NumericalInversion` method only computes scalar exponentials, but no logarithms. All other methods do not employ exponentials, but instead compute logarithms. Missing values ('–') denote cases where the method does not support the representation.

method (depending on the implementation one additional uniform random number may be required), since all of the required representations are Markovian. We therefore omit the method here.

Table 3.1 admits a number of observations. First, note that the costs for numerical inversion are independent of the structure of the representation and depend only on its size. In our example, this approach is by far the most efficient with respect to how many uniform random numbers are needed, but on the other hand requires 95 evaluations of the exponential function. Second, for characterisation methods that support both representations, the costs depend on the representation. With the Hyper-Erlang example, transformation to the CF-1 form increased both the worst-case and the average costs of the `Play` and `Count` methods. Furthermore, the `Count` method may actually be less efficient than generic `Play`, as the number of logarithms is determined by the size of the representation and not by its structure. Third, specialised methods help to reduce the costs. Due to the specialisation on chain structures, our `FE-diagonal` and `SimplePlay` methods are both more efficient than the generic methods, since by exploiting the structure they avoid drawing random numbers for the selection of successor states. `SimplePlay` is more efficient than `FE-diagonal` because it also utilises the fact that in bi-diagonal representations such as $(\boldsymbol{\alpha}', \mathbf{Q}')$ the feedback probability is zero, and therefore no geometric

Label	Operating System	libc/libm version	Hardware
M_1	Linux SUSE version 2.6.37.6-0.7-desktop	2.11.3	2 Intel Xeon 3.06 GHz, 6 GB RAM
M_2	Linux SUSE version 2.6.37.6-0.7-desktop	2.11.3	8 Dual-Core-AMD 8218 2.4 GHz, 32 GB RAM
M_3	Linux SUSE version 2.6.37.6-0.7-desktop	2.11.3	8 Quad-Core-AMD 8384 2.7 GHz, 64 GB RAM
M_4	Debian GNU/Linux 2.6.32-5-amd64	2.11.2	4 Dual-Core AMD Opteron 2212, 2.0 GHz, 32 GB RAM
M_5	Debian GNU/Linux 2.6.32-5-amd64	2.11.2	16 Intel Xeon X5550 Quad-Core, 2.67 GHz, 48 GB RAM
M_6	Debian GNU/Linux 2.6.37	2.7	1 Intel Pentium 4 Processor, 3.00 GHz, 2 GB RAM

Table 3.2: Machines used in the measurement study.

random variates are required. `eHFED`, `HFED` and `SimpleCount` reduce costs further by taking into account the branch structure of the representation. Not surprisingly, in this example the most efficient method is `SimpleCount`, which is also the most specialised of the algorithms discussed here, since it only supports Hyper-Erlang distributions.

3.7 Computational Costs of Atomic Operations

We augment our theoretical cost analysis by a study of the computational costs of the atomic operations. We use the number of uniform random numbers, #uni, and the number of logarithms, #ln, or the number of scalar exponentials, #exp, respectively, as cost metrics for random-variate generation. This study is motivated by the observation that the cost metrics are in general not identical, as most methods require more uniform random numbers than logarithms. With these methods, it can be advantageous to optimise such that the number of calls to the more expensive operation is minimised.

In this section we present a measurement study of the costs of atomic operations on typical systems to guide this optimisation. We use the following uniform number generator, as defined in [Jai91], p. 452:

$$u_{i+1} \quad = \quad \left(7^5 u_i\right) \mod \left(2^{31} - 1\right). \tag{3.67}$$

Equation (3.67) defines a typical random number generator, and, according to [Jai91], this generator provides desirable randomness and a large period length. We use the default `log()` and `exp()` operations of the C libraries on all systems. We performed 100 iterations, where in each step we called the respective operation $2 \cdot 10^8$ times, immediately discarding the result, and took timestamps before and after the step. The required time was then computed as the difference of both timestamps (the average time for one operation may be computed by dividing the difference by the number of operations). The code was optimised during compilation using the `-O3` option to the platform's C++ compiler (`g++`). Except for those applied by the compiler, no system-specific optimisations have been used. Each run was pinned to one CPU, in order to prevent process migration. Hardware and software data for the machines included in our measurement study are given in Table 3.2. Note that all of these run the Linux operating system. We considered using machines with Solaris, but due to the low clock resolution of 10 ms on these machines we could not obtain useful results.

Current POSIX systems offer several clocks that differ in accuracy and resolution. In order to determine the best clock for our experiments we ran the experiments with all of the four clocks specified in the POSIX standard ([IEE08b], `basedefs/time.h.html`). These clocks may differ not only in their resolution, but also in the degree to which they are affected by other processes running on the system. In the majority of our measurements, the `REALTIME` clock had the lowest squared coefficient of variation, which indicates that the measurements with this clock are not heavily influenced by other processes. As this clock is also supported on all of our machines and provides a resolution of 1 ns, we base our discussion on these results.

Results Figure 3.1 displays the minima of the run-times of 10^8 atomic operations, out of the 100 runs performed on each machine. We display the minima because these show the best-case run-times, and thus are least likely to have been affected by other user processes on the systems.[1] From these results we observe the following: First, the computational costs of the atomic operations differ greatly. On all machines logarithm operations take longer than the generation of a uniform random number; for instance, on M_4 `log()` is about four times slower than `uniform()`. With the exception of machines M_2 and M_3, the uniform random number generator is also faster than the `exp()` routine; on M_2 and M_4 `exp()` is slightly less expensive than `uniform()`. Second, the computational costs and the relations of the costs differ also between machines. In particular, on machines M_1 and M_6 `exp()` is approximately six times slower than `log()`, which in turn is more expensive than `uniform()`.

The differences in the measurement results may be caused by differences in the processor architecture, but a detailed investigation of the causes is out of scope for

[1] Virtually the same results have been obtained for the maxima, means, and medians.

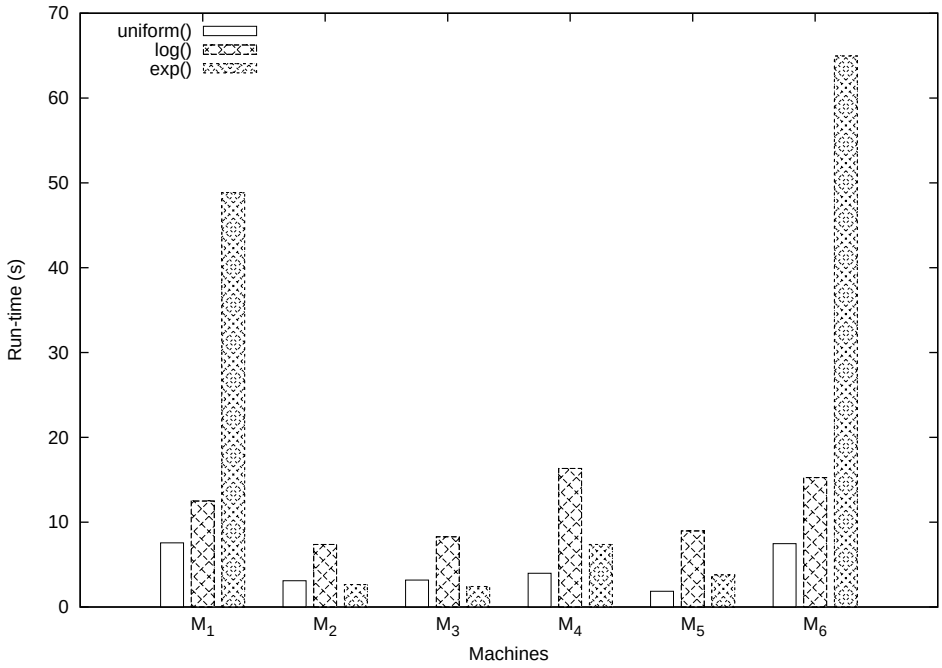


Figure 3.1: Minimal times for performing $2 \cdot 10^8$ atomic operations.

this work. Here, we note that the computation of an exponential and a logarithm are typically much more expensive than the generation of a uniform random number, if the uniform random-number generator from (3.67) is used. This is due to the fact that both the exponential and the logarithm are provided by complex software implementations in the systems' math libraries, while (3.67) can be implemented by relatively few operations.

When using characterisation methods based on the uniform random number generator from (3.67), we should aim to reduce the number of logarithms, both by appropriate choice of the representation and by choice of the method we apply. For instance, for our Hyper-Erlang example in Section 3.6, the `SimpleCount` method, which only requires a single logarithm, can be expected to be the most efficient. If we wanted to simulate general PH or acyclic PH distributions instead of Hyper-Erlang distributions, the `FE-diagonal` and `SimplePlay` methods, respectively, would be faster than the more generic `Play` and `Count` methods. In general, it appears that the `NumericalInversion` method is less efficient than any of the other algorithms considered here, at least if the scalar exponential is used to compute the matrix exponential. Even though computation of the exponential function may be slightly faster than generation of a uniform random number, the large number of exponentials that need to be computed outweighs this possible advantage. This also explains

the surprisingly bad performance observed in [HT11] for the inversion method using scalar exponentials.

It should be noted that the relation between the costs for atomic operations can be different for more complex random-number generators, for which uniform random numbers can be almost as expensive as logarithm operations [HRTW13]. In this case, the weighted cost metric defined in [HRTW13] can guide the choice of a suitable algorithm.

3.8 Concluding Remarks

This chapter provided a survey of algorithms for random-variate generation along with an analysis with respect to their computational costs. The main focus was on characterisation methods, which utilise the CTMC representation of a PH distribution. We proposed two new characterisation methods that use the canonical representations of the PH and APH distributions, respectively. The chapter gave four key insights that will be important for the remainder of this work:

First, the computational costs of of characterisation methods depend both on the representation and on the algorithm that is used. We saw that simpler representations and specialised methods can reduce the required number of uniforms and the number of logarithms.

Second, our measurement study shows that the costs for atomic operations differ significantly. In particular, logarithms and exponentials can be much more expensive than uniform random numbers, and therefore one should try to avoid these operations, even if this means that more uniform random numbers must be generated.

Third, the algorithms proposed for chain structures result in elegant expressions for the relationship between representations and cost metrics. With these expressions, it becomes possible to develop methods that optimise representations such that the costs are minimised. This will form the basis of the optimisation algorithms in the next chapter.

Fourth, suitable branch structures reduce the number of logarithms considerably. In particular, for Hyper-Erlang distributions the `SimpleCount` method requires only a single logarithm for each random variate. As we will show in Chapter 5, Hyper-Erlang distributions are also well-suited to fitting typical densities. Combined with our observation that they can be simulated efficiently, this makes them attractive for use in simulation studies.

We conclude this chapter by briefly touching upon the subject of accuracy. An algorithm for generating random variates from a PH distribution is considered accurate if the samples it generates follow the specified theoretical distribution. From a theoretical perspective, all of the characterisation methods can be expected to provide accurate results, because they draw random variates based directly on the

definition of PH distributions. Likewise, the acceptance/rejection method by [HT11] is also accurate, if it uses one of the characterisation methods. In practice, however, inaccuracies will be introduced by the implementation. Such issues can be caused by inaccuracies in the computation of the atomic operations as well as by inaccuracies in basic arithmetic that are due to the use of floating-point numbers. For instance, we already remarked in Section 3.2 that the product of a large set of small floating-point numbers in $(0, 1)$ (as required for the efficient generation of an Erlang sample) can appear to be zero if it is too small to be expressed as a non-zero floating-point number. The extent to which an implementation of one of the algorithms discussed here is affected by these inaccuracies depends on two major factors. First, details of the floating-point unit, the implementation of the atomic operations, and even of the compiler may affect the results (see [Mon08]). Second, the distribution itself and its representation will influence accuracy. In our example, simulation of Erlang distributions will be accurate up to a certain implementation-dependent length of the Erlang. Beyond this length, the product will be zero, and thus the logarithm operation will fail. As we remarked in Section 3.2, this problem may be addressed by splitting the product and computing several logarithms.

Due to the multitude of possible causes of inaccuracies, a formal comparison between the algorithms with respect to accuracy still remains an open question. However, our practical experience (see for instance the second case-study in Chapter 7) indicates that, except for extreme cases like in our example, the accuracy of random-variate generation using these algorithms is sufficiently high for practical use.

Chapter Four

Optimisation of PH Generators for Random-Variate Generation

The previous chapters introduced the basic formalism for phase-type (PH) distributions as well as algorithms for generating random variates from PH distributions. In Chapter 3 we observed that the costs of random-variate generation methods depend on the representation of the distribution. In this chapter we address the following optimisation problem:

> Starting from a Markovian representation of a PH distribution, find the (not necessarily minimal) Markovian representation that minimises the cost associated with generating random variates.

We first posed this research question in [RWBT09], but apart from the contributions in our own work, little progress has been made since then. In fact, there is only one other publication that touches upon the subject: In [HT11], Horváth and Telek describe transformation of a general PH distribution to extended Hyper-Feedback-Erlang (eHFED) form as part of the `AcceptanceRejection` method proposed in the paper. As illustrated in Chapter 3, the extended Hyper-Feedback-Erlang form may reduce the number of logarithms required. On the other hand, the eHFED representation for a general PH distribution may not be Markovian [HT11]. The `Acceptance/Rejection` method still allows drawing random variates from this representation using the `eHFED` method. This approach enables transformation of PH distributions to a representation that is known to be more efficient. On the other hand, it does not guarantee optimality with respect to the costs.

In this chapter we address the optimisation problem stated above using canonical forms for the acyclic phase-type (APH) and for the general PH classes. We use the canonical forms because every APH or PH distribution can be expressed in the respective form (see Section 2.4), and, therefore, our results then apply to the whole of these classes.

We proceed as follows: We first develop a general result for the optimum of APH distributions and then consider its generalisation to the whole PH class. In both parts we start with a representation in the respective canonical form (CF-1 for the APH case, Monocyclic for the PH case). Our optimisation procedures maintain the chain structure of the representations, i.e. the representation stays bi-diagonal in the APH case and FE-diagonal in the general PH case. This means that we can still apply the efficient methods for random-variate generation that we developed in the previous chapter.

Both parts employ the following observation from Chapter 3: Once entered, chain-structured representations have to be traversed completely, and thus the cost of random-variate generation depends on where the chain is entered. Consequently, by moving probability mass towards the absorbing state – 'to the right' –, the costs of random variate generation may be reduced. Note that, although not considered in the following, further optimisations may be possible if the number of states can be reduced before our methods are applied, as with smaller representations the lengths of paths through the CTMC may shrink. Reduction of the number of states may be achieved by e.g. eliminating obsolete poles of an APH distribution, as described in [Pul09].

4.1 Optimisation for Acyclic Phase-Type Distributions

In Chapter 3 we discussed algorithms for generating random variates from phase-type distributions. We observed that random variates from an acyclic phase-type (APH) distribution can be generated efficiently if the distribution is in a bi-diagonal representation (such as the CF-1 form), since such a representation does not require uniform random numbers for selection of successor states. With the `SimplePlay` method we observed that the number of logarithms and the number of uniforms both depend on the average number of state transitions up to absorption, for which we derived expression (3.36):

$$n^* \quad = \quad \boldsymbol{\alpha}\boldsymbol{\nu}^{\mathsf{T}}$$

where $\boldsymbol{\nu} = (n, n-1, \ldots, 1)$, or, equivalently

$$n^* \quad = \quad \sum_{i=1}^{n} \alpha_i (n - i + 1). \tag{4.1}$$

Apart from a constant offset of 1 additional random number which is required for initial state selection, the number of uniforms and the number of logarithms are equal (Equations (3.37) and (3.38)):

$$\#uni = n^* + 1 \quad \text{and} \quad \#ln = n^*.$$

These expressions hold for any bi-diagonal Markovian representation, since they do not rely on the ordering of the states. We try to solve the optimisation problem by finding a bi-diagonal representation $(\boldsymbol{\alpha}^*, \mathbf{Q}^*)$ for which the average number of traversed states,

$$n^*(\boldsymbol{\alpha}^*, \mathbf{Q}^*) \quad = \quad \boldsymbol{\alpha}^* \boldsymbol{\nu}^\mathsf{T} = \sum_{i=1}^{n} \alpha_i^*(n-i+1). \tag{4.2}$$

is minimal.

From the right side of (4.2) it is immediately obvious that, in order to achieve a reduction in n^*, probability mass must be shifted to the higher indices of the initial probability vector. At the same time, $(\boldsymbol{\alpha}^*, \mathbf{Q}^*)$ must represent the same distribution as $(\boldsymbol{\alpha}, \mathbf{Q})$. Informally, we apply the following idea: For $(\boldsymbol{\alpha}, \mathbf{Q})$ and $(\boldsymbol{\alpha}', \mathbf{Q}')$ to represent the same distribution, the Laplace-Stieltjes Transforms (LSTs) of the distributions must be the same. In particular, both LSTs must have the same poles. As the poles of the LST of a phase-type distribution are the eigenvalues of its sub-generator matrix (Theorem 2.1.2), $\mathbf{Q}$ and $\mathbf{Q}'$ must have the same eigenvalues. In a bi-diagonal form the eigenvalues are the entries of the diagonal. An operation that simply modifies the ordering of these entries without changing their values does not affect the poles of the LST. Changing the ordering of the diagonal entries requires modification of the initial vector as well. As discussed in Section 2.3.1, a new initialisation vector can be computed using a similarity transformation. As we will show, changes to the ordering of the entries of the diagonal can be expressed in terms of a similarity transformation. We thus consider shifting probability mass to the right by modifying the order of the rates along the diagonals.

We now give a formal description of the approach. We propose the following operator:

Definition 4.1.1. *The* `Swap(`$\boldsymbol{\alpha}, \mathbf{Q}, i$`)` operator *exchanges the ith rate with the $(i+1)$th rate ($1 \leq i \leq n-1$) on the diagonals in a bi-diagonal representation by swapping the ith and $(i+1)$th entry in the vector $\boldsymbol{\Lambda}$.*

Where appropriate, we also use the `Swap(`$\boldsymbol{\alpha}, \boldsymbol{\Lambda}, i$`)` *notation to denote the same operator.*

Lemma 4.1.1. *The* `Swap` *operator is described by the similarity transformation matrix $\mathbf{S}$:*

$$\mathbf{S} = \begin{pmatrix} \ddots & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & s_{i+1,i} & s_{i+1,i+1} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \ddots \end{pmatrix}$$

where

$$s_{i+1,i} = \frac{\lambda_i - \lambda_{i+1}}{\lambda_i}, \text{ and } s_{i+1,i+1} = \frac{\lambda_{i+1}}{\lambda_i} \text{ for } 1 \leq i \leq n-1.$$

Proof. From the definition it follows immediately that $\mathbf{S}\mathbb{1} = \mathbb{1}$ and that $\mathbf{S}$ is non-singular. Therefore, $\mathbf{S}$ is a similarity transformation matrix. Direct computation then shows that $\mathbf{Q}' = \mathbf{S}^{-1}\mathbf{Q}\mathbf{S}$ is the bi-diagonal matrix with the ith and $(i+1)$th entries of the diagonal exchanged. □

Remark 4.1.1. *`Swap`$(\boldsymbol{\alpha}, \boldsymbol{\Lambda}, i)$ only involves the ith and $(i+1)$th entries of $\boldsymbol{\alpha}$ and $\boldsymbol{\Lambda}$, which allows for the analytically tractable expressions employed in the following. Operators that produce more complex permutations in a single step do not have this property. Furthermore, repeated application of a pairwise exchange of adjacent entries in a list, as performed by the `Swap` operator, is sufficiently powerful to generate all permutations [Joh63].*

Let $(\boldsymbol{\alpha}', \mathbf{Q}')$ denote the result of applying `Swap`$(\boldsymbol{\alpha}, \mathbf{Q}, i)$ on $(\boldsymbol{\alpha}, \mathbf{Q})$. Because $(\boldsymbol{\alpha}', \mathbf{Q}')$ is derived by applying a similarity transformation to $(\boldsymbol{\alpha}, \mathbf{Q})$, both tuples represent the same distribution. Recall from Lemma 2.3.2 that

$$\boldsymbol{\alpha}' = \boldsymbol{\alpha}\mathbf{S}.$$

Then, the following properties of the result of the `Swap` operation are immediately obvious:

$$\forall j \neq i, i+1 : \alpha'_j = \alpha_j \tag{4.3}$$

$$\alpha'_i = \alpha_i + \alpha_{i+1}\frac{\lambda_i - \lambda_{i+1}}{\lambda_i} = \alpha_i + \alpha_{i+1}\left(1 - \frac{\lambda_{i+1}}{\lambda_i}\right) \tag{4.4}$$

$$\alpha'_{i+1} = \alpha_{i+1}\frac{\lambda_{i+1}}{\lambda_i} \tag{4.5}$$

$$n^*(\boldsymbol{\alpha}', \mathbf{Q}') = n^*(\boldsymbol{\alpha}, \mathbf{Q}) + \alpha_{i+1}\left(1 - \frac{\lambda_{i+1}}{\lambda_i}\right). \tag{4.6}$$

Equations (4.3)–(4.5) are valid for Markovian ($\boldsymbol{\alpha} \geq \mathbf{0}$) and non-Markovian ($\boldsymbol{\alpha} \in \mathbb{R}^n$) bi-diagonal representations. However, non-Markovian representations cannot be used in the `SimplePlay` method, since they do not admit the required state-space interpretation. As the cost definition based on the average number of traversed states is not valid for a non-Markovian representation, (4.6) is only applicable to Markovian representations.

If we restrict our attention to Markovian representations $(\boldsymbol{\alpha}, \mathbf{Q}), (\boldsymbol{\alpha}', \mathbf{Q}')$, we observe the following: The `Swap` operation with adjacent rates $\lambda_i < \lambda_{i+1}$, results in

$$n^*(\boldsymbol{\alpha}', \mathbf{Q}') \leq n^*(\boldsymbol{\alpha}, \mathbf{Q}) \tag{4.7}$$

according to (4.6), because from (4.4) and (4.5) we see that in this case

$$1 < \frac{\lambda_{i+1}}{\lambda_i}, \tag{4.8}$$

and α_{i+1} is non-negative. Consequently, by repeatedly exchanging adjacent rates $\lambda_i < \lambda_{i+1}$ such that the obtained representation is Markovian until no such operations are possible anymore, we can obtain a representation that has minimal costs n^*.

On the other hand, we note that the `Swap` operator will result in a non-stochastic vector $\boldsymbol{\alpha}'$ if

$$\alpha_i \quad < \quad \alpha_{i+1}\left(1 - \frac{\lambda_{i+1}}{\lambda_i}\right), \tag{4.9}$$

since then the resulting $\alpha'_i < 0$. In this case, $(\boldsymbol{\alpha}', \mathbf{Q}')$ is a non-Markovian representation. We must therefore avoid `Swap` operations that will result in non-stochastic $\boldsymbol{\alpha}'$. Based on these observations we propose the following

Theorem 4.1.1. *Given a Markovian representation $(\boldsymbol{\alpha}, \mathbf{Q})$ in CF-1 form, the representation $(\boldsymbol{\alpha}^*, \mathbf{Q}^*)$ that reverses the order of the rates is optimal with respect to n^* if $\boldsymbol{\alpha}^*$ is a stochastic vector. In this case, all bi-diagonal representations constructed by the `Swap` operator are Markovian.*

Proof. The proof consists of two steps. We first show the second part of the theorem, which states that all bi-diagonal representations are Markovian if $(\boldsymbol{\alpha}^*, \mathbf{Q}^*)$ is Markovian. We then show by contradiction that $(\boldsymbol{\alpha}^*, \mathbf{Q}^*)$ is optimal with respect to n^*.

According to property (4.4), a `Swap` operation applied to a Markovian representation can result in a non-Markovian representation only if a larger rate λ_{i+1} is exchanged for a smaller rate λ_i. Starting from $(\boldsymbol{\alpha}^*, \mathbf{Q}^*)$, all representations can be obtained by a series of `Swap` operations in which a smaller rate λ_{i+1} is exchanged for a larger rate λ_i. If $(\boldsymbol{\alpha}^*, \mathbf{Q}^*)$ is Markovian, none of these `Swap` operations can result in a non-Markovian representation.

For the proof of the first part of the theorem, recall that the CF-1 form is defined as a bi-diagonal form where the rates are in increasing order, i.e.

$$\lambda_1 \leq \cdots \leq \lambda_n. \tag{4.10}$$

The reversed CF-1 form then has the rates in decreasing order. Let us assume that $(\boldsymbol{\alpha}', \mathbf{Q}')$ with rates

$$\lambda'_1, \ldots, \lambda'_i, \lambda'_{i+1}, \ldots, \lambda'_n \tag{4.11}$$

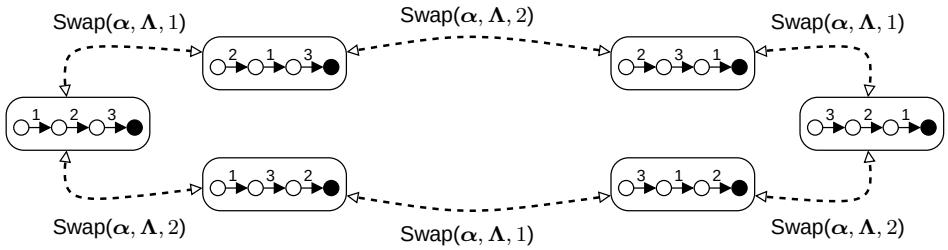


Figure 4.1: Search graph for optimisation on the bi-diagonal representation $(\boldsymbol{\alpha}, \boldsymbol{\Lambda})$, with rates $1, 2, 3$.

ordered such that $\lambda'_i < \lambda'_{i+1}$ is optimal. Then from (4.6) it follows that by exchanging $\lambda'_i, \lambda'_{i+1}$ using the `Swap` operator we can obtain a representation $(\boldsymbol{\alpha}'', \mathbf{Q}'') = \texttt{Swap}(\boldsymbol{\alpha}', \mathbf{Q}', i)$ for which $n^*(\boldsymbol{\alpha}'', \mathbf{Q}'') < n^*(\boldsymbol{\alpha}', \mathbf{Q}')$, i.e. we have a contradiction to our assumption that $(\boldsymbol{\alpha}', \mathbf{Q}')$ is optimal. □

4.1.1 Algorithms for Computing Optimal APH Representations

Theorem 4.1.1 states that if the reversed CF-1 form is Markovian, then this representation is optimal with respect to n^*. This optimal representation is easily computed by applying a similarity transformation. On the other hand, it is not guaranteed that the reversed CF-1 form is Markovian (see the examples in Section 4.1.2). In the following we develop algorithms for finding a Markovian APH representation that is close to optimal (with respect to n^*) when the reversed CF-1 form is non-Markovian.

The algorithms are best thought of as operating on a graph whose nodes are the permutations of the rate vector $\boldsymbol{\Lambda}$ and whose edges are defined by `Swap` operations, such that two nodes are connected if they can be transformed into each other by a single exchange of two adjacent rates (see Figure 4.1). From each permutation exactly $n - 1$ other permutations can be reached. If the reversed CF-1 is non-Markovian, then some of the permutations have non-Markovian initial vectors.

The most obvious approach proceeds by exploring the complete graph. This is equivalent to generating all permutations of $\boldsymbol{\Lambda}$ and minimising n^* over the subset of permutations whose initial vector is stochastic. The approach is easily implemented, e.g. as a modification to the Steinhaus-Johnson-Trotter algorithm for enumerating permutations [Joh63] and is guaranteed to find the optimum. Since this method explores all $n!$ permutations for an APH of size n, it is infeasible for larger APH.

Theorem 4.1.1 provides the basis for the two computationally less expensive algorithms presented in this section. The algorithms are more efficient than a simple enumeration because they avoid constructing non-Markovian representations. The underlying intuition is as follows: The optimal representation with respect to n^* is

somewhere on the Markovian side of the boundary between the Markovian and the non-Markovian representations. Theorem 4.1.1 implies that the Markovian optimum is along one of the paths from the CF-1 to the reversed CF-1. We thus need to find the (Markovian) point where the path between CF-1 and reversed CF-1 crosses the boundary between the Markovian and non-Markovian representations.

Our first algorithm, `BubbleSortOptimise` starts with the CF-1 form (i.e. inside the Markovian representations, at the left-most node in Figure 4.1). We know from (4.6) that each exchange of two adjacent rates such that after the exchange the larger rate is moved to the left constructs a new element of the path that has lower n^*, provided the new representation is Markovian. Properties (4.4) and (4.5) thus define the direction along which to search for the optimal Markovian representation without enumerating all permutations and without explicitly computing n^* for the new representation. `BubbleSortOptimise` follows from this intuition:

Algorithm `BubbleSortOptimise`$(\boldsymbol{\alpha}, \boldsymbol{\Lambda})$:
for $i = 1, \ldots, n-1$ **do**
  for $j = 1, \ldots, n-1$ **do**
    $(\boldsymbol{\alpha}', \boldsymbol{\Lambda}') :=$ `Swap`$(\boldsymbol{\alpha}, \boldsymbol{\Lambda}, i)$
    if $\boldsymbol{\Lambda}[j] < \boldsymbol{\Lambda}[j+1] \ \wedge \ \boldsymbol{\alpha}' \geq \mathbf{0}$ **then**
      $(\boldsymbol{\alpha}, \boldsymbol{\Lambda}) := (\boldsymbol{\alpha}', \boldsymbol{\Lambda}')$
    else
      break
    end if
  end for
end for
return $(\boldsymbol{\alpha}, \boldsymbol{\Lambda})$

The algorithm is a modified version of the popular `Bubblesort` algorithm [Knu97]. In general, the `Bubblesort` algorithm iterates repeatedly over a list and swaps all pairs of adjacent entries that are not in the desired order. This results in a global ordering after a finite number of iterations. The version presented here attempts to re-order the vector $\boldsymbol{\Lambda}$ given in CF-1 form into the reversed CF-1 form.

Note that `BubbleSortOptimise` does not perform `Swap` operations whose result would be non-Markovian, i.e. it does not cross the boundary between both types of representations. The algorithm terminates once either the reversed CF-1 form is reached or there are no re-orderings left that would result in a Markovian representation with lower cost n^*. While this may mean that the algorithm does not find Markovian representations hidden 'behind' non-Markovian ones, it is necessary because n^* has no meaning for non-Markovian representations.

Our second algorithm, `FindMarkovian`, starts from the reversed CF-1 form (the right-most node in Figure 4.1) and searches for the point where the path towards the CF-1 first crosses the border to the Markovian representations. The path is

constructed by swapping pairs of rates such that in the result the higher rate is to the right. That is, the algorithm attempts to re-order the reversed CF-1 form into the CF-1 form. The algorithm stops when it encounters a Markovian representation. Due to the fact that the CF-1 form is Markovian, the algorithm is guaranteed to terminate:

Algorithm `FindMarkovian`$(\boldsymbol{\alpha}, \boldsymbol{\Lambda})$:
Let $(\boldsymbol{\alpha}', \boldsymbol{\Lambda}')$ be the reversed CF-1 of $(\boldsymbol{\alpha}, \boldsymbol{\Lambda})$
while $\neg(\boldsymbol{\alpha}' \geq \mathbf{0})$ **do**
  $i := \operatorname{argmin}_i \{\alpha'_i < 0\}$
  $i := \max\{2, i\}$
  while $\neg(\boldsymbol{\alpha}' \geq \mathbf{0}) \wedge \exists k \ : \ \boldsymbol{\Lambda}[k] \geq \boldsymbol{\Lambda}[k+1]$ **do**
    $k := \operatorname{argmin}_j \{j \mid i-1 \leq j \leq n-1 \wedge \boldsymbol{\Lambda}[j] \geq \boldsymbol{\Lambda}[j+1]\}$
    $(\boldsymbol{\alpha}', \boldsymbol{\Lambda}') := \texttt{Swap}(\boldsymbol{\alpha}', \boldsymbol{\Lambda}', k)$
  end while
end while
return $(\boldsymbol{\alpha}', \boldsymbol{\Lambda}')$

Note that both algorithms test whether the theoretical optimum, i.e. the reversed CF-1 form (Theorem 4.1.1) is Markovian. This is guaranteed by the second part of Theorem 4.1.1 for `BubbleSortOptimise` and explicit in `FindMarkovian`. Both algorithms will therefore find the theoretical optimum from Theorem 4.1.1 if it is Markovian. On the other hand, neither algorithm is guaranteed to find the optimum if the reversed CF-1 form is not Markovian. `BubbleSortOptimise` might miss the optimum if finding the optimum requires constructing a non-Markovian representation. `FindMarkovian`, on the other hand, might not find the optimum because it stops when it reaches the first Markovian representation.

4.1.2 Illustrative Examples

A few examples will illustrate optimisation by re-ordering a bi-diagonal representation and the limits of this optimisation procedure. Furthermore, these examples will serve to evaluate the amount of cost reduction that can be achieved using the optimisation procedure proposed in this section. We first discuss three theoretical distributions that highlight advantages and limitations of the approach. We then study the approach on a distribution fitted to measurement data. In all examples we assume that the `SimplePlay` method will be used to generate random variates. The average costs are therefore given by $n^* = \boldsymbol{\alpha}\boldsymbol{\nu}^\mathsf{T}$ (cf. (3.36)).

Distribution with generalised Erlang structure. Consider the generalised Erlang distribution with $\boldsymbol{\Lambda} = (1, 2, 3, 4)$ and $\boldsymbol{\alpha} = (1, 0, 0, 0)$. For this distribution, every order of rates in $\boldsymbol{\Lambda}$ has costs $n^* = 4$, since no probability mass can be shifted to the

right without changing the distribution. As expected, both `BubblesortOptimise` and `FindMarkovian` identify $((1,0,0,0),(4,3,2,1))$ as the optimal representation.

APH with Markovian reversed CF-1. Let $\boldsymbol{\Lambda} = (1,2,3,4)$, as before, and the initial probability vector $\boldsymbol{\alpha} = (0.7, 0.15, 0.09, 0.06)$. Then, the average number of visited states is

$$n^*(\boldsymbol{\alpha}, \boldsymbol{\Lambda}) = 3.49.$$

Application of `BubblesortOptimise` results in the reversed CF-1 form with $\boldsymbol{\Lambda}' = (4,3,2,1)$, $\boldsymbol{\alpha}' = (0.46, 0.12, 0.18, 0.24)$ and average costs

$$n^*(\boldsymbol{\alpha}', \boldsymbol{\Lambda}') = 2.8.$$

Since the reversed CF-1 is Markovian, `FindMarkovian` gives the same result. We observe that probability mass in the initial probability vector has been shifted towards higher indices. In this case, the cost has been reduced by 19.7%.

APH with non-Markovian reversed CF-1. We study $(\boldsymbol{\alpha}, \boldsymbol{\Lambda})$ with $\boldsymbol{\Lambda} = (1,2,3,4)$ and $\boldsymbol{\alpha} = (0.5, 0.4, 0.05, 0.05)$. This representation has costs

$$n^*(\boldsymbol{\alpha}, \boldsymbol{\Lambda}) = 3.35.$$

The initial vector for the reversed CF-1 is $(-0.6, 1.4, 0, 0.2)$, and hence the reversed CF-1 form is non-Markovian. Applying the `BubblesortOptimise` algorithm to the CF-1 form provides us with a representation $(\boldsymbol{\alpha}', \boldsymbol{\Lambda}')$ with $\boldsymbol{\Lambda}' = (2,4,3,1)$ and $\boldsymbol{\alpha}' = (0.1, 0.7, 0, 0.2)$, for which

$$n^*(\boldsymbol{\alpha}', \boldsymbol{\Lambda}') = 2.7.$$

`FindMarkovian` starts on the non-Markovian reversed CF-1 representation and generates the Markovian representation $\boldsymbol{\Lambda}'' = (2,3,4,1)$ and $\boldsymbol{\alpha}'' = (0.1, 0.7, 0, 0.2)$, which has the same costs of 2.7. A complete enumeration of all permutations shows that both orderings are optimal with respect to n^*. The cost reduction with both algorithms is 19.4%.

APH fitted to empirical data. As the last example, we use an APH(8) fitted to the `loss1-50-opc-1` data-set from [RWW09] using the PhFit tool [HT02] (see also Chapter 5). This data set contains response-time measurements from a SOA system under high load and with network packet loss. The resulting APH has initial probability vector $\boldsymbol{\alpha} = (0.019, 0.006, 0.069, 0.104, 0.164, 0.371, 0.216, 0.051)$ and rate

vector

$$\begin{aligned}\boldsymbol{\Lambda} = (&7.181 \cdot 10^{-05}, 2.4280 \cdot 10^{-04}, 5.854 \cdot 10^{-04}, 5.863 \cdot 10^{-04},\\ &5.956 \cdot 10^{-04}, 5.965 \cdot 10^{-04}, 6.178 \cdot 10^{-04}, 6.332 \cdot 10^{-04}).\end{aligned}$$

For this representation,

$$n^*(\boldsymbol{\alpha}, \boldsymbol{\Lambda}) = 3.38.$$

Again, the reversed CF-1 for this representation has negative entries in the initial vector. Application of `BubblesortOptimise` results in $(\boldsymbol{\alpha}', \boldsymbol{\Lambda}')$ with initial probability vector $\boldsymbol{\alpha}' = (0.0047, 0.0203, 0.0614, 0.0929, 0.1327, 0.3911, 0.2417, 0.0552)$ and

$$\begin{aligned}\boldsymbol{\Lambda}' = (&2.4280 \cdot 10^{-04}, 7.181 \cdot 10^{-05}, 6.332 \cdot 10^{-04}, 6.178 \cdot 10^{-04},\\ &5.965 \cdot 10^{-04}, 5.956 \cdot 10^{-04}, 5.863 \cdot 10^{-04}, 5.854 \cdot 10^{-04}),\end{aligned}$$

which has $n^*(\boldsymbol{\alpha}', \boldsymbol{\Lambda}') = 3.256$. According to a complete enumeration, this is also the Markovian optimum. `FindMarkovian` returns

$$\boldsymbol{\alpha}'' = (0.0047, 0.0203, 0.069, 0.104, 0.164, 0.371, 0.216, 0.051)$$

and

$$\begin{aligned}\boldsymbol{\Lambda}'' = (&2.4280 \cdot 10^{-04}, 7.181 \cdot 10^{-05}, 5.854 \cdot 10^{-04}, 5.863 \cdot 10^{-04},\\ &5.956 \cdot 10^{-04}, 5.965 \cdot 10^{-04}, 6.178 \cdot 10^{-04}, 6.332 \cdot 10^{-04}),\end{aligned}$$

for which $n^*(\boldsymbol{\alpha}'', \boldsymbol{\Lambda}'') = 3.366$. With `BubbleSortOptimise` we thus achieve a cost reduction of 3.6%, whereas the representation returned by `FindMarkovian` is only 0.4% more efficient.

4.1.3 Discussion

In our examples we observe that the cost reduction obtained by the algorithms depends strongly on the given distribution. For distributions with (generalised) Erlang structure, optimisation by re-ordering of rates is not applicable. The same holds within blocks of subsequent phases with zero initial probability. For distributions where the probability mass is already concentrated at the higher indices in the CF-1, there is also little room for improvement.

In general, however, our examples indicate that re-ordering of rates can lead to significant cost reductions. While the improvement in efficiency depends on both the given distribution and on the algorithm, cost reductions can be as high as 19.7%. Furthermore, our last example shows that even with a typical APH distribution fitted to measurement data we can still achieve an improvement of 3.7%.

4.2 Optimisation for General Phase-type Distributions

The Monocyclic representation of the PH class is the generalisation of the CF-1 form for the APH class. The obvious similarity between both forms raises the question whether the result obtained in the previous section for the APH class can be generalised to the PH class in Monocyclic representation. In order to address this question, we first need to re-consider the underlying arguments of the discussion for APH distributions.

Unlike in the APH case, for PH distributions the expressions for the costs differ. The expression for the number of uniforms, $\#uni$ is rather complicated and does not provide us with a simple way of optimisation. However, the expression for the number of logarithms,

$$\#ln \quad = \quad 3\ell^* = 3\bar{\boldsymbol{\alpha}}\boldsymbol{\nu}^{\mathsf{T}}, \tag{4.12}$$

where $\bar{\boldsymbol{\alpha}}$ is the vector of block probabilities and

$$\boldsymbol{\nu} = (m, m-1, \ldots, 1) \tag{4.13}$$

is the number of blocks to be traversed is similar to that for the APH case, since it depends only on the number of visited FE blocks. We can apply our optimisation approach to this metric and try to minimise the number of logarithms. As logarithms are far more expensive than uniforms (as indicated in our measurement study in Chapter 3), this approach appears promising for improving efficiency. In the following, we therefore focus on optimisation for $\#ln$.

Generalising the approach for APH, we base our argument on the similarity transformation matrix $\mathbf{S}$ that swaps two adjacent FE blocks and produces the new initial probability vector $\boldsymbol{\alpha}' = \boldsymbol{\alpha}\mathbf{S}$. We generalise the definition of the `Swap` operator as follows:

Definition 4.2.1. *The* `GSwap`$(\boldsymbol{\alpha}, \mathbf{Q}, i)$ operator *exchanges the ith FE block with the $(i+1)$th FE block ($1 \leq i \leq m-1$) on the diagonal in a block-bi-diagonal representation by swapping the ith and $(i+1)$th entry in the vector $\boldsymbol{\Upsilon}$ (or, equivalently, by swapping the block matrices associated with the FE blocks in the sub-generator). The associated similarity transformation matrix $\mathbf{S}$ has the form*

$$\mathbf{S} = \begin{pmatrix} \mathbf{I}_{\nu\times\nu} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{S}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_{\mu\times\mu} \end{pmatrix},$$

where

$$\nu = \sum_{k=1}^{i-1} b_k, \ \mu = \sum_{k=i+2}^{m} b_k,$$

are the number of states in front of and behind the two blocks, respectively, and $\hat{\mathbf{S}} \in \mathbb{R}^{(b_i+b_{i+1})\times(b_i+b_{i+1})}$ *is the lower-triangular matrix that describes the effect of the similarity transformation on the two Feedback-Erlang blocks.* $\hat{\mathbf{S}}$ *is the solution of*

$$\begin{pmatrix} \mathbf{F}_i & -\mathbf{F}_i \mathbb{1}\!\!\mathbb{I}\mathbf{e}_1 \\ \mathbf{0} & \mathbf{F}_{i+1} \end{pmatrix} \hat{\mathbf{S}} = \hat{\mathbf{S}} \begin{pmatrix} \mathbf{F}_{i+1} & -\mathbf{F}_{i+1} \mathbb{1}\!\!\mathbb{I}\mathbf{e}_1 \\ \mathbf{0} & \mathbf{F}_i \end{pmatrix} \tag{4.14}$$

$$\hat{\mathbf{S}} \mathbb{1}\!\!\mathbb{I} = \mathbb{1}\!\!\mathbb{I}. \tag{4.15}$$

Due to the block upper-triangular structure of the coefficient matrices in (4.14), $\hat{\mathbf{S}}$ is a block lower-triangular matrix. We use the $\texttt{GSwap}(\boldsymbol{\alpha}, \boldsymbol{\Upsilon}, i)$ notation to denote the swap operation of the ith and $(i+1)$th FE blocks.

We summarise the properties of the `GSwap` operator in the following remarks. While the `GSwap` operator is a generalisation of the `Swap` operator, the fact that the FE-diagonal form is block-bi-diagonal with block matrices potentially having size larger than 1 results in one important difference, described in Remark 4.2.2.

Remark 4.2.1. *The structure of* $\mathbf{S}$*, which describes the* `GSwap` *operator (i.e.* $\mathbf{Q}' = \mathbf{S}^{-1}\mathbf{Q}\mathbf{S}$*), ensures that* `GSwap` *has only local effects on the initial probability vector, i.e. it only affects the entries of the initial probability vector that belong to the two FE blocks being swapped. We denote the vector of these probabilities by* $\hat{\boldsymbol{\alpha}}$*.*

Remark 4.2.2. *Given two Feedback-Erlang blocks* (b_i, λ_i, z_i) *and* $(b_{i+1}, \lambda_{i+1}, z_{i+1})$ *with corresponding sub-generator matrix*

$$\mathbf{Q} = \begin{pmatrix} \ddots & \ddots & & & & & \\ & -\lambda_i & \lambda_i & & & & \\ & & \ddots & \ddots & & & \\ & z_i\lambda_i & & -\lambda_i & (1-z_i)\lambda_i & & \\ & & & & -\lambda_{i+1} & \lambda_{i+1} & \\ & & & & & \ddots & \ddots \\ & & & & z_{i+1}\lambda_{i+1} & & -\lambda_{i+1} \\ & & & & & & \ddots \end{pmatrix},$$

application of the `GSwap` *operator exchanges the entire block matrices corresponding*

to the FE blocks instead of individual rates along the diagonal, resulting in

$$\mathbf{Q}' = \begin{pmatrix} \ddots & \ddots & & & & & & \\ & -\lambda_{i+1} & \lambda_{i+1} & & & & & \\ & & \ddots & \ddots & & & & \\ & z_{i+1}\lambda_{i+1} & & -\lambda_{i+1} & (1-z_{i+1})\lambda_{i+1} & & & \\ & & & & -\lambda_i & \lambda_i & & \\ & & & & & \ddots & \ddots & \\ & & & & z_i\lambda_i & & -\lambda_i & (1-z_i)\lambda_i \\ & & & & & & \ddots & \ddots \end{pmatrix}.$$

This means that the `GSwap` *operator maintains the proper exit rates. If $z_i > 0$ or $z_{i+1} > 0$, the operation cannot be broken down into exchanges of individual rates, since any such exchange would modify the FE blocks, and thus alter the eigenvalues of the matrix, producing a different phase-type distribution.*

Remark 4.2.3. *The Steinhaus-Johnson-Trotter theorem [Joh63] ensures that all permutations of FE blocks can be obtained by repeated application of the* `GSwap` *operator.*

According to Theorem 4.1.1, for acyclic phase-type distributions in bi-diagonal form any swap of transition rates that moves a phase with a higher rate away from the absorbing state and maintains a Markovian initialisation vector results in a more efficient representation for simulation. This means that for APH distributions the direction of search for the optimal representation is known solely from the properties of the sub-generator matrix and it is independent of the initial probability vector. The corresponding statement for FE-diagonal forms would be straightforward, using FE blocks and the reversed Monocyclic form. Unfortunately, this statement does not hold for PH distributions in FE-diagonal form, as we demonstrate with the following counterexample:

Example 4.2.1. *Let* $\mathbf{Q}$ *denote the Monocyclic sub-generator matrix defined by* $\boldsymbol{\Upsilon} = ((1, 0.1, 0), (b_1, \lambda_1, z_1), (b_2, \lambda_2, z_2))$*, where*

$$b_1 = 3, \lambda_1 = 1.5, z_1 = 0.5 \text{ and } b_2 = 3, \lambda_2 = 1, z_2 = 0.$$

Exchanging the second and third block results in $\boldsymbol{\Upsilon}' = ((1, 0.1, 0), (b_2, \lambda_2, z_2), (b_1, \lambda_1, z_1))$ *and the associated sub-generator matrix* $\mathbf{Q}'$*. Let*

$$\mathbf{S} = \begin{pmatrix} 1 & \\ & \hat{\mathbf{S}} \end{pmatrix} \quad \text{such that} \quad \mathbf{Q}' = \mathbf{S}^{-1}\mathbf{Q}\mathbf{S} \tag{4.16}$$

denote the similarity transformation that describes the `GSwap` *operation for this case, i.e.* $\hat{\mathbf{S}}$ *is the solution of Equations 4.14 and 4.15:*

$$\hat{\mathbf{S}} = \begin{pmatrix} 1 & & & & & \\ 0.3333 & 0.6667 & & & & \\ 0.1111 & 0.4444 & 0.4444 & & & \\ -0.925926 & 0.4444 & 0.8889 & 0.592593 & & \\ 0 & -0.925926 & 0.4444 & 0.592593 & 0.8889 & \\ 0 & 0 & -0.925926 & 0.148148 & 0.4444 & 1.3333 \end{pmatrix} \tag{4.17}$$

Now consider the two initial vectors

$$\boldsymbol{\alpha}_1 = (0.09 \mid 0.1, 0.3, 0.31 \mid 0.1, 0.1, 0),$$

and

$$\boldsymbol{\alpha}_2 = (0.09 \mid 0.1, 0.3, 0.31 \mid 0.2, 0, 0)$$

whose only difference is the distribution of the probability mass assigned to the third FE block (block boundaries are indicated by $\mid$*). The costs for random-variate generation from* $(\boldsymbol{\alpha}_1, \mathbf{Q})$ *and* $(\boldsymbol{\alpha}_2, \mathbf{Q})$ *are (Equation (3.33)):*

$$\#ln_{(\boldsymbol{\alpha}_1,\mathbf{Q})} = \#ln_{(\boldsymbol{\alpha}_2,\mathbf{Q})} = 3 \cdot (0.09 \cdot 3 + 0.71 \cdot 2 + 0.2) = 5.67.$$

After swapping the two blocks using $\mathbf{S}$ *the resulting initial probability vectors are*

$$\boldsymbol{\alpha}_1' = \boldsymbol{\alpha}_1\mathbf{S} = (0.09 \mid 0.141852, 0.28963, 0.271111 \mid 0.118519, 0.0888889, 0)$$

and

$$\boldsymbol{\alpha}_2' = \boldsymbol{\alpha}_2\mathbf{S} = (0.09 \mid 0.0492593, 0.426667, 0.315556 \mid 0.118519, 0, 0)$$

respectively. Note that in $\boldsymbol{\alpha}_1'$ *the initial probability mass assigned to the third FE block increased from* 0.2 *to* 0.207407*, while in* $\boldsymbol{\alpha}_2'$ *the probability mass for this block decreased from* 0.2 *to* 0.118519*. The costs of random-variate generation changed as follows:*

$$\begin{aligned} \#ln_{(\boldsymbol{\alpha}_1',\mathbf{Q}')} &= 3 \cdot 1.8825939 = 5.6477817, \\ \#ln_{(\boldsymbol{\alpha}_2',\mathbf{Q}')} &= 3 \cdot 1.9714836 = 5.9144508. \end{aligned}$$

That is, with $\boldsymbol{\alpha}_1$ *swapping the blocks resulted in a cost decrease, while with* $\boldsymbol{\alpha}_2$ *costs increased.*

Complete enumeration of all orderings shows that $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_1'$ and $\boldsymbol{\alpha}_2, \boldsymbol{\alpha}_2'$ are the only Markovian representations for this example. The example thus illustrates that with

true FE-diagonal representations (i.e. those with non-degenerate FE blocks) the effect of swapping two consecutive FE blocks may depend not only on the properties of the sub-generator of the distribution, but also on the distribution of the mass in the initial probability vector. Consequently, Theorem 4.1.1 cannot be generalised to the whole PH class.

4.3 Algorithms for Monocyclic Optimisation

Example 4.2.1 shows that the reversed Monocyclic form of a general PH distribution with non-degenerate FE-diagonal representation is not guaranteed to be optimal, even if the initialisation vector is Markovian. Hence the efficient optimisation methods developed for the APH class cannot be applied to the PH class, since, first, there is no general optimum that only needs to be checked for non-negativity of the initialisation vector, and, second, the direction in which to search for the optimum cannot be derived from the sub-generator alone. On the other hand, Example 4.2.1 is not just bad news, since it also shows that a cost reduction by swapping adjacent FE blocks is indeed possible, even though the effects of swapping two blocks are not predictable.

As with the CF-1 form, the optimum for the Monocyclic form can be found by an optimisation of the costs over the set of all permutations of the Feedback-Erlang blocks. This approach is guaranteed to find the optimum with respect to all permutations, but involves generating and checking $m!$ representations. The approach may be feasible if neither the number of FE blocks m nor the block lengths are too large, but running-times become prohibitive for large m or large FE blocks. Large m require a large number of permutations to be checked, while large FE blocks imply large $\hat{\mathbf{S}}$, and thus higher costs for computing the initial probability vector $\hat{\boldsymbol{\alpha}}'$ resulting from each `GSwap` operation.

Therefore, we propose extensions of the algorithms for efficient APH optimisation to the Monocyclic case by developing heuristics that replace the strict ordering criterion available for the CF-1 case. In both algorithms, the comparison of two adjacent rates must be replaced by a call to the generic routine `ComparisonHeuristic`, which returns `true` or `false`, depending on whether the two Feedback-Erlang blocks given as its arguments are in the order imposed by the heuristic. The extended version `GBubbleSortOptimise` is as follows:

Algorithm `GBubbleSortOptimise`$(\boldsymbol{\alpha}, \boldsymbol{\Upsilon})$:
for $i = 1, \ldots, m-1$ **do**
  for $j = 1, \ldots, m-1$ **do**
    $(\boldsymbol{\alpha}', \boldsymbol{\Upsilon}') :=$`Swap`$(\boldsymbol{\alpha}, \boldsymbol{\Upsilon}, i)$
    if `ComparisonHeuristic`$(\boldsymbol{\alpha}, \boldsymbol{\Upsilon}, j) = \text{true} \wedge \boldsymbol{\alpha}' \geq \mathbf{0}$ **then**
      $(\boldsymbol{\alpha}, \boldsymbol{\Upsilon}) := (\boldsymbol{\alpha}', \boldsymbol{\Upsilon}')$

else
break
end if
end for
end for
return $(\boldsymbol{\alpha}, \boldsymbol{\Upsilon})$

The `GBubbleSortOptimise` algorithm is guaranteed to always find a Markovian representation with any heuristic, since it does not leave the region of orderings with Markovian initialisation vectors.

The `GFindMarkovian` algorithm, on the other hand, starts with a possibly non-Markovian representation and is guaranteed to find a Markovian representation only with the eigenvalues heuristic discussed below. The eigenvalues heuristic imposes decreasing order of the dominant eigenvalues of the Feedback-Erlang blocks. With this heuristic, `GFindMarkovian` will in the worst case terminate with the original Markovian Monocyclic representation. In order to guarantee termination of the algorithm with other heuristics that do not have this property, we terminate if all $m!$ possible orderings have been evaluated. In this case, the algorithm may return a non-Markovian representation. The algorithm is given in the following:

Algorithm `GFindMarkovian`$(\boldsymbol{\alpha}, \boldsymbol{\Upsilon})$`:`
Let $(\boldsymbol{\alpha}', \boldsymbol{\Upsilon}')$ be the reversed Monocyclic form of $(\boldsymbol{\alpha}, \boldsymbol{\Upsilon}')$
r:=0
while $\neg(\boldsymbol{\alpha}' \geq \mathbf{0})$ **do**
$i := \text{argmin}_i \{\alpha'_i < 0\}$
$i := \max\{2, i\}$
while $\neg(\boldsymbol{\alpha}' \geq \mathbf{0}) \wedge \exists k$: `ComparisonHeuristic`$(\boldsymbol{\Upsilon}[k], \boldsymbol{\Upsilon}[k+1]) =$ false **do**
$k := \text{argmin}_j \{j \mid i-1 \leq j \leq m-1 \wedge \boldsymbol{\Upsilon}[j] \geq \boldsymbol{\Upsilon}[j+1]\}$
$(\boldsymbol{\alpha}', \boldsymbol{\Upsilon}') :=$ `Swap`$(\boldsymbol{\alpha}', \boldsymbol{\Upsilon}', k)$
if $(\boldsymbol{\alpha}', \boldsymbol{\Upsilon}')$ is a new representation **then**
$r++$
end if
if $r = m!$ **then**
goto END
end if
end while
end while
END:
return $(\boldsymbol{\alpha}', \boldsymbol{\Upsilon}')$

4.3.1 Heuristics for Optimisation

In the following we present four heuristics that can be used as `ComparisonHeuristic` in either algorithm. The heuristics presented here have been derived based on the following argument: In Theorem 4.1.1 we showed that the optimal ordering for the APH case is obtained if the ordering of the elements of the diagonal of the CF-1 form is reversed (provided that this ordering is Markovian). Due to the simplicity of the CF-1 form, this re-ordering can be seen equivalently as a re-ordering of the dominant eigenvalues, of the means, and of the exit rates. Furthermore, property (4.5) corresponds to the determinant of the swap matrix being larger than 1. These four criteria differ from each other if true Feedback-Erlang blocks are compared.

Eigenvalues Heuristic The eigenvalues heuristic relates to the CF-1 case most directly. Recall that the Monocyclic representation is defined such that the Feedback Erlang blocks are ordered along the diagonal according to increasing absolute value of their dominant eigenvalues. The eigenvalues heuristic directly applies the observation from the CF-1 case that swapping two blocks may move probability mass to the right iff the eigenvalue of the right phase is larger than that of the left phase. Equivalently, in the FE-diagonal case probability mass may be moved to the right if the dominant eigenvalue of the right FE block is larger than that of the left FE block.

Definition 4.3.1 (Eigenvalues heuristic)**.** *Let* $(\boldsymbol{\alpha}, \boldsymbol{\Upsilon})$ *be a FE-diagonal representation and let* r_i, r_{i+1} *be the dominant eigenvalues of the* i*th and* $(i+1)$*th block, respectively. Then, the* eigenvalues heuristic *is defined as*

$$\texttt{EigenvaluesHeuristic}(\boldsymbol{\alpha}, \boldsymbol{\Upsilon}, i) \;:=\; \begin{cases} \mathit{true} & |r_i| < |r_{i+1}|\,, \\ \mathit{false} & \mathit{else}, \end{cases} \tag{4.18}$$

that is, the heuristic returns true if the absolute value of the dominant eigenvalue of the i*th block is larger than that of the* $(i+1)$*th block.*

Mean Heuristic The mean heuristic stems from the following observation for the bi-diagonal case: Re-ordering phases directly relates to re-ordering the means of the associated distributions. I.e., swapping two phases such that the one with higher rate is moved to the left is equivalent to swapping them such that the lower mean moves to the left. This idea can be applied to the FE-diagonal case as follows. First, we have to assign a mean to each FE block. While this is unambiguous in the bi-diagonal case (where FE blocks are of length 1), in the Monocyclic case probability mass may be assigned to all phases of a block, rendering the mean dependent on the distribution of the mass. The most straightforward approach is then to assign probability mass of 1 to the first entry of the block. The mean of a Feedback-Erlang

block (b_i, λ_i, z_i) with sub-generator matrix $\mathbf{F}_i$ and probability mass 1 at the first entry is

$$\hat{M}_i \quad = \quad \mathbf{e}_1(-\mathbf{F}_i)^{-1}\mathbb{1}. \tag{4.19}$$

The mean of a Feedback-Erlang block (b_i, λ_i, z_i) with sub-generator matrix $\mathbf{F}_i$ and initial probability vector $\boldsymbol{\alpha}_i$ is

$$M_i = \frac{\boldsymbol{\alpha}_i}{\boldsymbol{\alpha}_i\mathbb{1}}(-\mathbf{F}_i)^{-1}\mathbb{1}. \tag{4.20}$$

Definition 4.3.2 (Mean heuristic)**.** *Let* $(\boldsymbol{\alpha}, \boldsymbol{\Upsilon})$ *be an FE-diagonal representation and let* $\hat{M}_i, \hat{M}_{i+1}$ *be the mean of the* i*th and* $(i+1)$*th Feedback-Erlang block, when starting at the first entry of each block. Then,*

$$\texttt{MeanHeuristic}(\boldsymbol{\alpha}, \boldsymbol{\Upsilon}, i) \quad := \quad \begin{cases} \textit{true} & \hat{M}_i > \hat{M}_{i+1} \\ \textit{false} & \textit{else}, \end{cases} \tag{4.21}$$

defines the mean heuristic*. A different version of the heuristic can be defined by using the means* M_i, M_{i+1} *of the distributions defined by the FE blocks, as given in (4.20).*

Exit-Rates Heuristic The exit-rates heuristic is another generalisation of an observation from the bi-diagonal case. Exit rates are an indicator for the time spent in a Feedback-Erlang block or a phase, respectively. In the bi-diagonal case, the exit rates are equal to the rates of the phases, and moving the larger rate to the left is equivalent to moving the higher exit rate. For an FE-diagonal representation, the exit rates of the ith and $(i+1)$th block are given by $(1-z_i)\lambda_i$ and $(1-z_{i+1})\lambda_{i+1}$. Based on the results for the bi-diagonal case, optimisation consists in re-ordering blocks such that the highest exit rates (i.e. largest rates λ_i with a bi-diagonal representation) move to the left.

Definition 4.3.3 (Exit-rates heuristic)**.** *Let* $(\boldsymbol{\alpha}, \boldsymbol{\Upsilon})$ *be an FE-diagonal representation with exit rates* $(1-z_i)\lambda_i$ *and* $(1-z_{i+1})\lambda_{i+1}$*. The* exit-rates heuristic is defined as

$$\texttt{ExitRatesHeuristic}(\boldsymbol{\alpha}, \boldsymbol{\Upsilon}, i) \quad := \quad \begin{cases} \textit{true} & (1-z_i)\lambda_i < (1-z_{i+1})\lambda_{i+1}, \\ \textit{false} & \textit{else}. \end{cases} \tag{4.22}$$

	$\mathbf{F}_1$	$\mathbf{F}_2$
Dominant eigenvalue	$r_1 = -0.3095$	$r_2 = -1$
Mean with mass 1 at first state	$\hat{M}_1 = 4$	$\hat{M}_2 = 3$
Mean with normalised mass ($\boldsymbol{\alpha}_1$)	$M_1 = 4$	$M_2 = 1.7042$
Mean with normalised mass ($\boldsymbol{\alpha}_2$)	$M_1 = 2.5$	$M_2 = 1.7042$
Exit rate	0.75	1

Table 4.1: Properties of the FE blocks in Example 4.2.1.

Determinant Heuristic In Lemma 4.1.1 we showed that for the APH case the similarity transformation matrix $\hat{\mathbf{S}}$ has the following explicit structure:

$$\hat{\mathbf{S}} = \begin{pmatrix} 1 & 0 \\ \frac{\lambda_i - \lambda_{i+1}}{\lambda_i} & \frac{\lambda_{i+1}}{\lambda_i}, \end{pmatrix} \tag{4.23}$$

with determinant $\det(\hat{\mathbf{S}}) = {}^{\lambda_{i+1}}/_{\lambda_i}$. In this case, swapping two adjacent rates moves probability mass to the right iff $\lambda_{i+1} > \lambda_i$, or, equivalently, if $\det(\hat{\mathbf{S}}) > 1$. The determinant heuristic generalises this criterion to the general case.

Definition 4.3.4 (Determinant heuristic)**.** *Let* $(\boldsymbol{\alpha}, \boldsymbol{\Upsilon})$ *be an FE-diagonal representation and let* $\hat{\mathbf{S}}$ *be the similarity transformation matrix that exchanges the* i*th and* $(i+1)$*th block. The* determinant comparison heuristic *is defined as*

$$\texttt{DeterminantHeuristic}(\boldsymbol{\alpha}, \boldsymbol{\Upsilon}, i) := \begin{cases} \textit{true} & \det(\hat{\mathbf{S}}) > 1, \\ \textit{false} & \textit{else}. \end{cases} \tag{4.24}$$

Discussion While these heuristics are exact for degenerate FE blocks, they may be misleading with non-degenerate FE blocks, as can be illustrated using Example 4.2.1. Table 4.1 shows the relevant properties considered by the eigenvalues, mean, and exit rates heuristics. The determinant of the swap matrix $\hat{\mathbf{S}}$ is $|\hat{\mathbf{S}}| = 0.208$. Observe that the eigenvalues, mean, and exit rates heuristics would recommend to swap the two blocks. As we saw in the counterexample, this is correct for $\boldsymbol{\alpha}_1$, but incorrect for $\boldsymbol{\alpha}_2$. Likewise, the prediction by the determinant heuristic that swapping would not move probability mass to the right is wrong for $\boldsymbol{\alpha}_1$, but correct for $\boldsymbol{\alpha}_2$.

4.3.2 Application

As noted in the above discussion, for FE-diagonal representations with non-degenerate FE blocks the heuristics can give a correct prediction of the effect of swapping two blocks, but none of the heuristics are *guaranteed* to move probability

mass to the right. Therefore, the orderings returned by `GBubbleSortOptimise` and `GFindMarkovian` are not guaranteed to be optimal with respect to random variate generation. On the other hand, the counterexample only affects non-degenerate FE blocks. If an FE-diagonal representation contains degenerate FE-blocks, we may expect these to be returned in the optimal order for the APH case. One would typically run the algorithms with several heuristics and pick the best result.

4.4 Evaluation

As we have observed throughout this chapter, the cost reduction obtainable by optimisation through reordering depends on the distribution itself. In this section we provide an empirical study on the applicability and effectiveness of optimisation with arbitrary distributions.

Our experiment proceeds as follows: For size n and number of FE blocks m we randomly generate N Monocyclic distributions $F_1, \ldots, F_N$ by creating a random Markovian initialisation vector $\boldsymbol{\alpha}$ and a random ordered FE-block list $\boldsymbol{\Upsilon}$ of length m such that the overall size of the representation is n. For each distribution F_i we then compute all $m!$ representations $(\boldsymbol{\alpha}_{i,j}, \boldsymbol{\Upsilon}_{i,j})$ where, for $j = 1, \ldots, m!$, $\boldsymbol{\Upsilon}_{i,j}$ is a permutation of $\boldsymbol{\Upsilon}$ and $\boldsymbol{\alpha}_{i,j}$ is the initial vector corresponding to this distribution (obtained by a similarity transformation). From these, we choose the Markovian initialisation vector $\boldsymbol{\alpha}'_i$ that has the lowest number of traversed blocks, ℓ^*. We can then compute the cost reduction for distribution F_i,

$$\Delta\ell_i^* \quad = \quad \ell^*(\boldsymbol{\alpha}_i, \boldsymbol{\Upsilon}_i) - \ell^*(\boldsymbol{\alpha}'_i, \boldsymbol{\Upsilon}'_i) \tag{4.25}$$

In our experiment, we set $N = 5001$, $n = 7$, and let m run from $2, \ldots, 7$. Note that $m = n = 7$ represents the case of APH distributions in bi-diagonal form. The results of the experiment are shown in Table 4.2. The table shows the improvement probability, defined as the ratio between the number of distributions for which an improvement was observed and the number of distributions tried, and the mean, variance and maximum improvement for the distributions where re-ordering yielded a cost reduction. We first observe that successful optimisation becomes more likely with higher numbers of blocks m (and consequently lower block lengths). One explanation is that with long blocks many re-orderings give non-Markovian initialisation vectors. Second, we note that the average savings in cases where optimisation was possible are around 0.35 for all m, i.e. if an optimisation is possible, on average 0.35 block traversals (or, equivalently, about 1 logarithm) can be saved. Furthermore, we observe that maximum absolute improvements vary between 1.72299 and 2.27931 block traversals.

While such an experiment must necessarily be incomplete, as it cannot provide an exhaustive search of the complete set of phase-type distributions, our results

m	Improvement probability	Mean($\Delta\ell^*$)	Var($\Delta\ell^*$)	Max($\Delta\ell^*$)
2	0	–	–	–
3	0.5957	0.361612	0.098564	1.72299
4	0.8776	0.357491	0.122194	2.26036
5	0.9732	0.354051	0.110254	2.27931
6	0.9988	0.344626	0.0694888	1.75109
7	1	0.355659	0.0555498	1.79303

Table 4.2: Average and maximum optimisation results for randomly-generated Monocyclic generators.

illustrate that even with randomly chosen distributions optimisation by re-ordering is possible and can save up to 2.27931 block traversals. This further supports our earlier observation for the bi-diagonal case that significant cost improvements are possible.

4.5 Concluding Remarks

In this chapter we studied optimisation of random-variate generation from phase-type distributions using chain-structured representations. For the sub-class of acyclic phase-type distributions in bi-diagonal representation we showed that the reverse CF-1 form minimises the costs of random-variate generation, both in terms of logarithms and in terms of uniform random numbers, if this representation is Markovian. Even for distributions whose reversed CF-1 form is not Markovian, an optimal representation can be found by starting from the CF-1 form and moving higher rates towards the start of the distribution. This general optimisation criterion is completely independent of the initial probability vector.

A counterexample told us that there is no equivalent result for distributions from the general PH class in FE-diagonal form. With FE-diagonal representations containing true Feedback-Erlang blocks, the effect of moving the blocks depends mainly on the initial vector, and therefore an optimality result based on the generator cannot be provided.

Based on these insights, we developed two algorithms for optimisation of bi-diagonal and FE-diagonal representations. The algorithms for the APH class search for the optimal ordering based on the optimality result. The algorithms for the general PH class extend the algorithms for the APH class by utilising one of four proposed heuristics for predicting the effect of exchanging Feedback-Erlang blocks. While these heuristics may be misleading for FE-diagonal representations with true FE-blocks, they directly translate to the same exact criterion for re-ordering rates in the bi-diagonal case if the distribution is a acyclic.

An empirical study with random phase-type distributions complemented our theoretical discussion by illustrating that cost improvements are indeed possible for many distributions.

Chapter Five

Efficient and User-Friendly PH Fitting

The main advantage of using phase-type distributions instead of other statistical distributions in system evaluation is their ability to capture important statistical properties of empirical data sets, such as empirical moments or the shape of the density. Fitting phase-type distributions to measurement data has received a lot of attention in recent years, and several methods for fitting data have been developed, along with tools that make the algorithms available in practice. Approaches for fitting PH distributions to data include moment-matching, non-linear optimisation of a distance function, and expectation-maximisation (EM) algorithms that maximise likelihood.

In this chapter we first develop a new fitting approach based on partitioning of the data set. Partitioning of the data set prior to fitting can improve fitting quality. In particular, the family of segmentation-based approaches aims to capture oscillations in the density as well as heavy-tailed behaviour by splitting the data set into segments with low variability, which can be fitted by simple distributions using the EM algorithm (see Section 5.2). We propose the use of clustering to detect important features of the density and to partition the data set such that clusters contain samples belonging to the same feature. Clusters are fitted by distributions with a simple structure, which then form the branches of the overall PH model. This approach yields results that fit the density well even with data sets that are difficult to fit by a PH distribution. It produces mixtures of distributions, which enable efficient algorithms for random-variate generation (see Chapter 3). Furthermore, with clustering the user can control the quality of the fit by setting initial cluster centers on a plot of the histogram, which is a very intuitive way of adjusting parameters of the fitting algorithm.

Clustering constitutes the basis of the HyperStar tool. The tool has a graphical user interface and supports various algorithms for fitting the clusters. It can easily

be extended to provide a GUI for prototypes of new fitting algorithms. The name of the tool follows from its general approach: HyperStar fits mixtures of phase-type distributions, such as Hyper-exponential or Hyper-Erlang distributions.

In the second part of the chapter we describe an algorithm for fitting FE-diagonal representations. Our evaluation of a prototype implementation of the algorithm using HyperStar shows that the algorithm may provide good fitting results with few parameters and little user interaction.

The chapter is structured as follows: We first describe our cluster-based approach to PH fitting. Section 5.2 places the algorithm in the context of other methods, with a particular focus on approaches that partition the data prior to fitting. Section 5.3 gives an overview of our implementation of cluster-based fitting in the extensible HyperStar tool. We evaluate the method in Section 5.4 with three typical data sets, employing standard quality measures for PH distributions, such as log-likelihood values and moment errors to assess fitting quality. We provide a comparison to previous partitioning approaches as well as to two standard tools for PH fitting. In Section 5.5 we illustrate how HyperStar can aid the development of new fitting algorithms, using a new algorithm for fitting FE-diagonal representations.

5.1 Cluster-based Fitting

Our cluster-based fitting approach consists of splitting the input data into M clusters and fitting each cluster with a phase-type distribution. The distributions fitting the individual clusters are then combined in a branch-structured phase-type distribution. More formally, the complete sample set $S = \{s_1, \ldots, s_N\}$ is split into M clusters $C_1, \ldots, C_M$ such that each sample belongs to exactly one cluster. The samples in each cluster C_i are then fitted by a phase-type distribution $(\boldsymbol{\alpha}^{(i)}, \mathbf{Q}^{(i)})$. The class and fitting method for this distribution may be chosen arbitrarily. After the fitting process for the clusters is completed, the complete distribution $(\boldsymbol{\alpha}, \mathbf{Q})$ is obtained as a mixture of the cluster distributions by weighting the initial probabilities according to relative cluster size:

$$\boldsymbol{\alpha} := \left(\frac{|C_1|}{N}\boldsymbol{\alpha}^{(1)}, \ldots, \frac{|C_M|}{N}\boldsymbol{\alpha}^{(M)} \right) \tag{5.1}$$

and constructing a branch-structured sub-generator matrix

$$\mathbf{Q} := \begin{pmatrix} \mathbf{Q}^{(1)} & \mathbf{0} & & \\ & \ddots & \ddots & \\ & & & \mathbf{0} \\ & & & \mathbf{Q}^{(M)} \end{pmatrix}. \tag{5.2}$$

We propose two-step clustering: First, k-means clustering [Llo82] is used to quickly identify clusters, and second, the initial clustering is refined by re-assigning samples to clusters based on the distributions fitted to the clusters.

5.1.1 Initial Clustering

The initial clustering uses the simple k-means algorithm [Llo82]: Given M initial cluster centers $c_1, \ldots, c_m$, each sample s_i is first assigned to the closest cluster C_j:

$$s_i \in C_j \quad \Leftrightarrow \quad j = \operatorname*{argmin}_j \{|c_j - s_i| \ : \ j = 1, \ldots, M\}. \tag{5.3}$$

Then, each cluster center c_j is updated to be the mean of the samples associated to the cluster C_j:

$$c_j \quad := \quad \frac{1}{|C_j|} \sum_{i \in \mathcal{I}_j} s_i, \tag{5.4}$$

where $\mathcal{I}_j := \{i \ : \ s_i \in C_j\}$ denotes the indices of the samples assigned to the jth cluster. Both steps are repeated alternatingly until the cluster centers stabilise. The user specifies the number of the clusters and the position of the initial cluster centers by marking important peaks of the density in the GUI.

5.1.2 Cluster Refinement

The cluster refinement step combines the fitting of PH distributions to the clusters and the refinement of the sample assignments to clusters, as follows: In each iteration, we first fit a PH distribution $(\boldsymbol{\alpha}^{(j)}, \mathbf{Q}^{(j)})$ to each cluster C_j using the fitting module selected for that cluster. We then re-assign samples to clusters using one of the two re-assignment strategies described below. Both steps are repeated until either convergence occurs or a maximal number of iterations has been exceeded. We check convergence by comparing the parameters of the branch distributions. Note that this criterion is only guaranteed to detect convergence for canonical representations of the branch distributions.

The most obvious strategy for cluster refinement is to assign each sample to the cluster whose density is highest for that sample:

$$s_i \in C_j \quad \Leftrightarrow \quad j = \operatorname*{argmax}_j \left\{ \hat{f}_j(s_i) \ : \ j = 1, \ldots, M \right\}, \tag{5.5}$$

where $\hat{f}_j(t) = \boldsymbol{\alpha}^{(j)} \mathrm{e}^{\mathbf{Q}^{(j)} t} (-\mathbf{Q}^{(j)} \mathbb{1})$ denotes the density of the phase-type distribution fitted to the jth cluster. While this strategy is efficient and often yields good fitting

results, it tends to create clusters with low variance, particularly if the branch distributions are of Erlang type. Such clusters then necessitate a large number of phases to be represented well (cf. [AS87]). This issue may be addressed by the probabilistic re-assignment strategy: For each sample we compute the vector

$$\boldsymbol{\pi} \;:=\; \frac{1}{\sum_{j=1}^{M} \hat{f}_j(s_i)} \left(\hat{f}_1(s_i), \ldots, \hat{f}_M(s_i) \right) \tag{5.6}$$

which, for each cluster C_j, estimates the probability that the sample s_i is in this cluster. We then draw a discrete random variate j' from the distribution defined by $\boldsymbol{\pi}$ and assign the sample s_i to cluster $C_{j'}$. Unlike the default strategy, the probabilistic strategy is not guaranteed to converge and requires the specification of a maximum number of iterations after which the cluster assignment should be accepted. Additionally, due to the additional computations, it is less efficient than the default strategy. On the other hand, the probabilistic re-assignment strategy increases the variance inside the clusters and thus allows the fitting of clusters with smaller distributions.

5.2 Related Work

There exists a large number of approaches for fitting PH distributions to data. Attempts at gaining a structured overview of the field have been presented in e.g. [LA96] and [MZ09a, MZ09b], but, due to the diverse directions being followed in ongoing research work, it appears that any general survey must necessarily be incomplete and will quickly become outdated. Surveys of a sub-area, which are often presented in the 'Related Work' sections of papers on specific approaches like the ones discussed in the following, are usually more complete, but limited to that area. In the following we provide a brief overview of existing approaches and then discuss the relation between these and our cluster-based method.

Fitting approaches for PH distributions differ with respect to the methods they apply. Moment-matching (e.g. [BPdL98, BHT05, TH02a, CZS08]) aims to fit the moments of the empirical distribution, typically using explicit analytical expressions for the moments of the PH distribution to derive the parameters of the distribution. Moment-matching fits the moments of the distribution well, but often does not capture the shape of the density or the distribution function (see e.g. observations in [FW98, RW08b, Dan10]). The quantile-matching approach of Feldmann and Whitt [FW98] uses an iterative procedure to obtain a hyper-exponential distribution that fits the higher quantiles of the distribution well and is thus especially suited to fitting long-tailed distributions. As noted in [FW98], the approach cannot fit distributions with non-monotone densities. Optimisation methods (e.g. [MR93, BT94, Fad98, HT02, IH05]) fit PH distributions iteratively by optimis-

ing a distance function such as the density area distance, maximum likelihood, or entropy. These methods are well-suited to obtaining good fits of the shape of the distribution, but may not fit the moments exactly. Several authors have used variants of the expectation-maximisation (EM) algorithm to fit the whole PH class [ANO96] or sub-classes [RDS02, SH08, WZXL05, TBT06, WLS08]. Depending on the subclass that is used, the EM algorithm can give good a fitting for the shape of the distribution [TBT06] or of its higher quantiles [RDS02, SH08, WZXL05, WLS08].

Our focus in this chapter is on cluster-based fitting. Although partitioning of the data set has been proposed before, the idea of a clustering algorithm for identification of important features of the density has not been applied to PH distributions before.

The most similar approach is the family of segmentation-based EM approaches described in [RDS02, SH08, WZXL05, WLS08]. These approaches divide the data set into partitions with a specified maximum coefficient of variation and then fit hyper-exponential [RDS02, SH08] or Hyper-Erlang [WZXL05, WLS08] distributions to the partitions using the EM algorithm. We summarise the approach as given in [WLS08]: The samples are first sorted in increasing order. Starting from the lowes samples, the ordered data set is then successively split into segments with a pre-specified maximum coefficient of variation cv. A Hyper-Erlang distribution is fitted to each segment separately using the AEM and BEM algorithms. The BEM algorithm performs a fixed number of iterations to fit a Hyper-Erlang distribution of a given size to a set of samples. The size of the distribution is specified by the number of branches. All branches have the same length, and the length depends on the number of branches by a linear factor k. The AEM algorithm computes the optimal size of a Hyper-Erlang distribution by repeatedly calling the BEM algorithm with increasing numbers of branches until the rates for the two most similar branches differ by less than a given threshold ϵ. We will refer to the combination of splitting and fitting by AEM and BEM by the acronym SEM. We are not aware of any tools implementing the SEM algorithm.

With this family of segmentation-based algorithms, the goal of partitioning is to reduce variability in the individual partitions, in order to be able to capture heavy-tailed behaviour of the data set [RDS02]. The extension to Hyper-Erlang distributions in [WZXL05, WLS08] aims to also fit peaks in the density [WZXL05, WLS08], which is not possible with hyper-exponential distributions [FW98]. With cluster-based fitting, on the other hand, our primary goal is to detect and fit peaks and other important features of the density. Such peaks can often be approximated well by simple distributions like the Erlang distribution. As we will illustrate in the evaluation, the approach of partitioning the data based on a constant maximal coefficient of variation may yield good results with suitable data sets and a good threshold. On the other hand, the fitting quality is very sensitive to the choice of the threshold, and threshold selection such that important features are fitted well is

difficult. In contrast, our cluster-based fitting approach is more flexible in detecting similar values, as it determines the size of clusters automatically.

k-means clustering has been proposed as a method for MMPP (Markov-Modulated Poisson Process) fitting by [GHH03]. This approach uses clustering only to obtain initial values for an EM algorithm. Another idea similar to clustering is followed in G-FIT [TBT06], which fits Hyper-Erlang distributions using the Expectation-Maximisation (EM) algorithm. G-FIT alternates between assigning samples to branches and adjusting the Erlang parameters to fit the assigned samples. Sample assignments in G-FIT are 'soft', in that each sample has a certain probability of belonging to every branch. That is, samples do not belong to one branch exclusively. In contrast, in the cluster-based approach we develop here, sample assignments are 'hard': Each sample belongs to exactly one cluster and is fitted by exactly one branch. This promises several advantages: First, as clusters are independent data sets, cluster fitting may be performed independently and the data sets belonging to each cluster are smaller than the whole data set. This offers performance benefits. Second, with independent clusters, each cluster may be fitted by different distributions and different fitting methods. This allows easy integration of existing methods, as well as new methods. Note that hard sample-assignment as used in our algorithm has the disadvantage that it may yield distributions with a very large number of phases, in particular if there are many clusters or if the variance within clusters is very low. Large distributions may have only limited applicability in further evaluation. In particular, analytical approaches typically suffer state-space explosion if the distribution is too large. On the other hand, in simulation even large distributions may still be acceptable, if they have a simple structure.

The PhFit tool [HT02] fits APH distributions in CF-1 form. The tool can fit the body and tail part of the distributions with different algorithms, if the user specifies the start of the tail. Although we had limited success in body/tail fitting, we observed that even without explicit tail fitting PhFit often yields good results. Body fitting with PhFit uses a variant of the Frank/Wolfe method [FW56] for optimisation of a non-linear goal function. In this application, the goal function is a distance measure between the empirical and the fitted distribution. It is minimised by local linearisation and a search in the direction of steepest descent. PhFit supports the density distance, CDF distance, and entropy as distance functions.

5.3 Implementation

Cluster-based fitting forms the core of the HyperStar tool. HyperStar was developed to be user-friendly and provide good-fitting results, with a particular emphasis on using the fitted distributions in simulation. In order to provide good fitting results and enable fast random-variate generation, the tool supports branch structures, where each branch is fitted to a cluster of the input data. We have im-

plemented Hyper-Erlang, Hyper-Exponential-Erlang, and Hyper-Feedback-Erlang fitting for the branches, as these distributions enable efficient fitting and efficient use in simulation. The tool also supports the use of other fitting methods, including PhFit [HT02], for fitting the branches. The tool was designed around a graphical user interface that allows the selection of important features of the empirical data, such that the fitting algorithms can focus on representing these features in the phase-type distribution. Furthermore, HyperStar provides immediate graphical feedback of the fitting results. Due to its generic graphical user interface and extendability, HyperStar may also be used to quickly provide a GUI to experimental fitting methods implemented in e.g. Mathematica. We provide an overview of HyperStar here; implementation details are available in [Kra12].

There are several other approaches that address usability and interfacing to existing fitting tools in different ways. The jphase tool [PR06] implements algorithms from existing tools such as G-FIT and PhFit and augments them with a unified graphical user interface. Similarly, the ProFiDo tool [BBK10] integrates existing fitting tools for PH distributions and stochastic processes into workflows using a graphical user interface and an XML exchange format. The KPC-Toolbox [CZS08] offers a set of Matlab routines for automatically fitting Markovian Arrival Processes and PH distributions, matching a user-specified set of moments with a PH distribution. With respect to usability, HyperStar extends the facilities provided by these tools by allowing the user to optimise fitting results by intuitive selection of important features of the distribution. Unlike ProFiDo, HyperStar does not support workflows, but provides a graphical user interface especially suited for prototype implementations.

5.3.1 Architecture of the HyperStar Tool

The general architecture of HyperStar is shown in Figure 5.1. The HyperStar kernel calls the other modules, provides communication between them, and implements the graphical user interface. The GUI shows a histogram and the empirical cumulative density function of the data. The user may explore the data by e.g. zooming into the plot, selecting histogram bar widths, or truncating the data set. After interesting features of the data (such as peaks in the density) have been identified, they can be marked. The clustering algorithms use these marks as initial cluster centers. Each cluster is fitted by one branch of the distribution. With some fitting modules, the user may additionally indicate important features by adding weights to intervals. Fitting modules can be selected and parameterised through the GUI. In the following, we give a short summary of the modules:

Import modules read data from a data source (e.g. a file) and convert it to HyperStar's internal format. Currently, only data files are supported, but e.g. SQL modules may give access to data that is stored in a database.

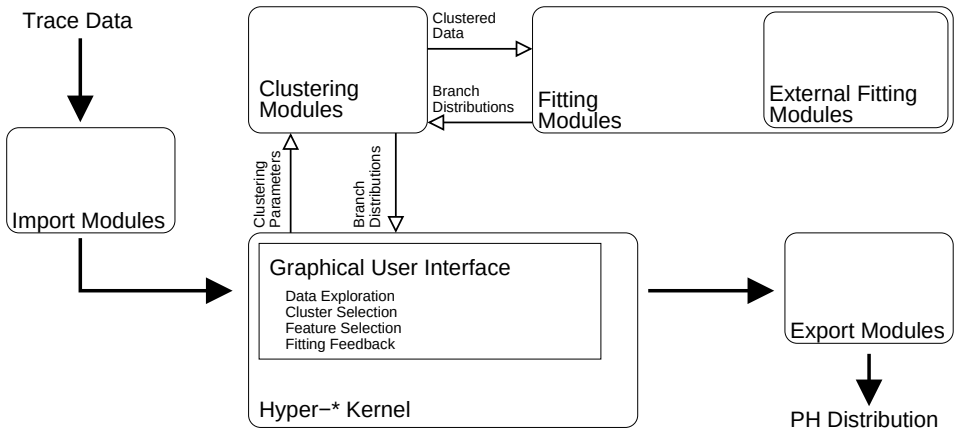


Figure 5.1: Architecture of the HyperStar tool.

Clustering Modules	k-Means Clustering; Distribution-based cluster refinement; Probabilistic distribution-based cluster refinement; Data segmentation [WLS08]
Internal Fitting Modules	Erlang fitting (uses moment-matching); Exponential-Erlang/Erlang-Exponential Fitting [BHT05]; Exponential (uses moment-matching); AEM/BEM Hyper-Erlang fitting [WLS08]
External Fitting Modules	PhFit [HT02]; G-FIT [TBT06]; Mathematica Interface
Import Modules	Text files with one sample per line (PhFit input format); Text files with one sample per line, starting with the number of samples (G-FIT input format)
Export Modules	Mathematica script; R[R D06] script; XML; Libphprng format [BBH$^+$11]

Table 5.1: HyperStar Modules

Clustering Modules are parameterised by the initial cluster center selections of the user. Clustering modules directly call the assigned fitting module and use the results for refinement. The clustering modules return a fitted distribution to the HyperStar kernel.

Fitting Modules implement the fitting of distributions to data. Internal fitting modules implement fitting algorithms directly as Java classes, while external fitting modules provide an interface to established tools, such as G-FIT or PhFit, and to prototype implementations in Mathematica. Internal fitting modules (see Table 5.1)

have been selected such that they produce simple branch distributions. The Mathematica interface enables the user to quickly add HyperStar's graphical user interface to new fitting algorithms. Each fitting module returns a distribution fitted to the input data.

Export Modules export the fitted distribution in a variety of data formats that can then be read by e.g. libraries for random-variate generation (see Chapter 7 and [RH12, BBH+11]).

5.3.2 Special Features

HyperStar can be parameterised in several ways to fulfil the requirements of different application scenarios. While not exhaustive, the following list summarises a few of the supported features.

Manual Mode. The user may set the number of initial clustering steps and the number of refinement steps to 1. In this mode, the cluster centers are not adjusted, and no refinement is performed. This mode allows the greatest control over the fitting process.

Pure k-Means Clustering. When the number of initial clustering steps is larger than 1 and the number of refinement steps is 1, HyperStar applies k-means clustering but does not refine the clusters based on the fitted distributions.

Pure Distribution-based Clustering. In contrast, by setting the number of initial clustering steps to 1 and the number of refinement steps above 1, HyperStar can be used to only perform clustering according to the distribution-based re-assignment strategies described in the previous section.

Default Mode: Clustering and Refinement. In the default mode, both the number of initial clustering steps and the number of refinement steps are larger than 1. In general, this mode produces good results without requiring much user interaction.

GUI Mode. By setting the number of clusters to 1 and requesting only 1 initial clustering and cluster refinement step, the whole data set is passed into the fitting module. HyperStar then acts as a GUI to the fitting module.

5.4 Evaluation

We evaluate the quality of cluster-based fitting using the HyperStar tool by studying three data sets whose densities and distributions exhibit typical properties of real-world phenomena. We first give an overview of the data sets. We then evaluate fitting quality with cluster-based fitting and compare it to G-FIT and PhFit, before providing a comparison of our algorithm and the segmentation based approach.

Our first data set, S_1, consists of 10^6 samples from a 20-phase APH distribution fitted to measurements in a SOA testbed with fault injection (cf. [RW10]). We use this distribution for two reasons. First, the shape of the density of this distribution

	Size	Mean	Variance	c^2
S_1	9505	3.521	14.197	1.1452
S_2	1864	0.59465	0.145145	0.410465
S_3	99946	4896.70	16014128	0.667878

Table 5.2: Statistical properties of the data sets used in the evaluation.

(Figure 5.2), with its pronounced peak at 3 s, is typical for response-times with TCP connections over unreliable links. Second, we use an APH fitted to the data rather than the data itself in order to judge the fitting quality in a case where it is known that a PH distribution does exist.

The second data set, S_2, contains samples of the packet delivery ratios in the DES-Testbed [BGJ+11], a testbed that implements a wireless mesh network. Each sample gives the average packet delivery ratio of a different link in this network. This data set is particularly interesting because the range of samples is limited to $[0, 1]$, with density equal to zero beyond 1. As PH distributions have density $f(t) > 0$ for $t > 0$, such a density cannot be fitted exactly. Still, a good fit of the particular shape of the density on $[0, 1]$ is required, and no standard statistical distribution with limited support provides such a fit. A PH distribution may be fitted in an automated way to reflect the density well on $[0, 1]$. A typical way of using this distribution in simulation would then be truncation at 1. We use this density here to evaluate how well the fitting approaches fit non-PH densities.

Our last data set, S_3, contains samples for the packet transmission delay of PTP packets in a highly-detailed simulation of a network router (cf. [WRM11] and the application example in Section 7.3). The most important aspect of this data set is that the low quantiles must be fitted well, while the higher quantiles are not as important. Furthermore, the data set may be difficult to fit, as it consists of a large set of samples equal to zero and a set of nearly uniformly-distributed samples above zero. Statistical properties of the data sets are summarised in Table 5.2.

We evaluate fitting quality using plots of the density or distribution function. This visual assessment is complemented by the common quantitative measures for PH fitting quality given in [LA96] and summarised in Table 5.3. These measures describe how well the fitted distribution captures the moments, the distribution and the density of the data. Additionally, we compute the log-likelihood of the fitted PH distribution, and the absolute error in the 1% quantile for one of our data sets. In general, the choice of a fitting result based on these quality criteria will depend on the further use of the distribution. For instance, for an analysis using queueing theory, good fitting of the first three moments is sufficient. On the other hand, simulation may require other criteria, such as a good fit of the density. Our S_3 data set provides an instance where the further use of the model in simulation requires

Quality Measure	Definition
Area difference between distribution functions	$\Delta F = \int_0^\infty \lvert\hat{F}(t) - F(t)\rvert dt$
Area difference between densities	$\Delta f = \int_0^\infty \lvert\hat{f}(t) - f(t)\rvert dt$
Log-likelihood	$\mathcal{L} = \sum_{i=1}^{N} \ln \hat{f}(x_i)$
Relative error in the first moment	$e_1 = \frac{\lvert\hat{c}_1 - c_1\rvert}{c_1}$
Relative error in the second central moment	$e_2 = \frac{\lvert\hat{c}_2 - c_2\rvert}{c_1}$
Relative error in the third central moment	$e_3 = \frac{\lvert\hat{c}_3 - c_3\rvert}{c_3}$
Absolute error in the 1% quantile	$e_q = \hat{F}^{-1}(0.01) - F^{-1}(0.01)$

Table 5.3: PH-fitting quality measures, partly based on [LA96].

Method	n	Δf	ΔF	$\mathcal{L}$	e_1	e_2	e_3
k-Means	21	9.5	6.6	-19381.83	0.007	0.031	0.034
100 Iterations	21	7.4	4.8	-19239.85	5×10^{-15}	0.0198	0.0109
Prob. Assignment	20	6.2	2.0	-19186.92	0.002	0.007	0.0227
G-FIT	20	3.5	1.4	-19035.68	9×10^{-15}	0.006	0.003
PhFit	20	5.0	8.1	-19146.52	0.048	0.0127	0.239
SEM ($cv = 0.1$)	56	22.03	39.7	-20534.84	0.0005	0.4737	1.820
SEM ($cv = 0.5$)	52	14.36	12.4	-19635.71	0.0005	0.0506	0.170
SEM ($cv = 0.6$)	16	18.22	10.9	-20840.48	0.0005	0.0448	0.126
SEM ($cv = 1.01$)	18	18.82	6.74	-22630.55	0.0005	0.0082	0.053

Table 5.4: Quality measures for the S_1 data set.

a good fit of the 1% quantile. Definitions for the quantitative measures are shown in Table 5.3. Except for the log-likelihood, all of these measures have an optimum at zero, i.e. for a fitted distribution that fits the empirical distribution perfectly, these measures would be zero. For the log-likelihood, higher values indicate better fitting quality. We compute the density-area difference measure by building a histogram of 1000 equi-spaced buckets and sampling the fitted density at the bucket boundaries.

5.4.1 Comparison of Cluster-based Fitting to PhFit and G-FIT

We apply cluster-based Hyper-Erlang fitting using HyperStar and vary the clustering parameters: We use pure k-means clustering, k-means clustering with 100 refinement steps, and k-means clustering with 100 refinement steps and probabilistic cluster assignments. We first compare the fitting results to those of G-FIT and of PhFit. We parameterise branch lengths for G-FIT similarly to the branch lengths obtained

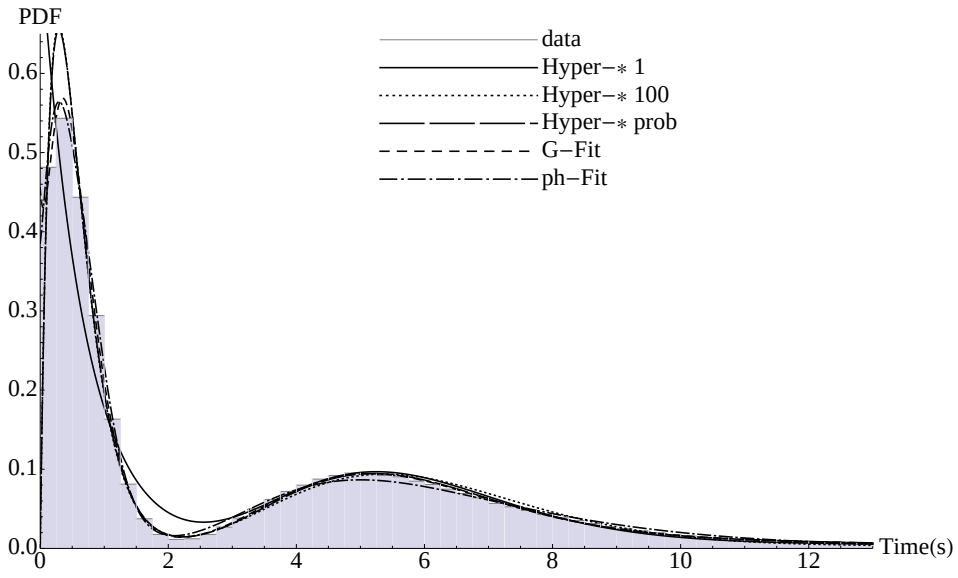


Figure 5.2: Histogram of the S_1 data set and densities of fitted distributions.

with our tool. For PhFit we set the size of the APH distribution such that it is similar to our tool, but still provides good results.

Figure 5.2 shows the histogram of the S_1 data set and the densities of the distributions fitted with HyperStar, G-FIT and PhFit. As this data set is from an APH distribution, we expect the fitting tools to be able to approximate it well.

Visually, both G-FIT and PhFit give good results. Hyper-Erlang fitting using simple k-means clustering without cluster refinement misplaces the first peak and neglects the first dent. Cluster-refinement using either the density-based or the probabilistic strategy captures the dent well and places the first peak correctly, but still overestimates this peak slightly. The visual results are corroborated by the quantitative measures, shown in Table 5.4: All fitting tools give distributions of similar size and with similar quality measures. The table also illustrates that G-FIT gave the best match for the first three moments and the highest log-likelihood for this data set.

Fitting results for the second data set are shown in Figure 5.3. As this data set has limited support, we know that it cannot be fitted accurately by a PH distribution. However, from the application where it was obtained we know that the peaks closte to 0, 0.75 and 1 are important characteristics of the data set that must be reflected in an approximating distribution.

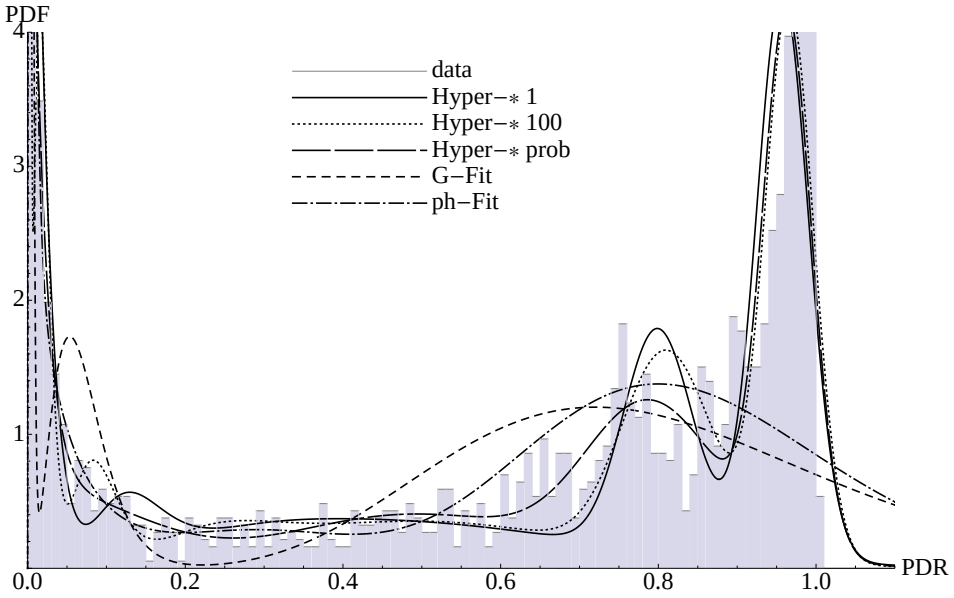


Figure 5.3: Histogram of the S_2 data set and densities of fitted distributions.

Neither G-FIT nor PhFit provided a good fit for these features. G-FIT places an additional peak around 0.75 and does not represent the peaks at higher values. Furthermore, it exaggerates the valley between 0.1 and 0.75. PhFit fits the start of the empirical density well, but also neglects the other two peaks. In contrast, our cluster-based algorithm gave good fittings for this density, irrespective of the parameterisation. The quantitative evaluation results in Table 5.5 show that all tools fitted the first three moments well, but that clustering gave lower area distance errors and higher log-likelihood values. Note, however, that clustering also used a much higher number of phases than either G-FIT or PhFit. While more phases might have improved results especially with G-FIT, we were not able to test this assumption, since G-FIT did not return valid results with branch lengths above 200 (possibly due to numerical errors). Branch fitting using our approach did not suffer from such scalability issues. The results in Table 5.5 also illustrate that probabilistic cluster-assignments may reduce the number of phases without sacrificing fitting quality. In fact, according to the quantitative measures, probabilistic assignment gave the best result, although visual inspection indicates that k-means and density-based cluster refinement capture the second peak better.

The last data set, S_3, contains a large amount of samples equal to zero and may therefore be difficult to fit by an algorithm that assumes that no mass is present at zero. One way to obtain a PH distribution for such a data set is to fit a distribution

Method	n	Δf	ΔF	$\mathcal{L}$	e_1	e_2	e_3
k-Means	1215	576	14.4	854.74	0.007	0.0136	0.024
100 Iterations	1233	587	11.9	975.90	2×10^{-15}	0.008	0.017
Prob. Assignment	1121	542	5.7	1034.28	0.004	0.005	0.004
G-FIT	433	631	48.1	113.06	1×10^{-14}	0.033	0.124
PhFit	20	464	31.4	469.20	0.012	0.004	0.049
SEM ($cv = 0.1$)	36	733.9	125.6	-344.50	0.0004	0.4636	1.802
SEM ($cv = 0.5$)	18	1612.8	584.3	-7232.33	0.9806	0.9920	0.987
SEM ($cv = 0.6$)	16	941.1	116.5	-1201.56	0.0004	0.0168	0.105
SEM ($cv = 1.01$)	8	840.5	118.5	-4510.97	0.2822	0.4624	0.9101

Table 5.5: Quality measures for the S_2 data set.

to the non-zero portion and then rescale the entries of the initial vector such that the resulting distribution has an appropriate point mass at zero. While some tools might give better results when used in this way, it requires manual intervention and is not particularly intuitive. We therefore only consider fitting of the complete data set. The non-zero portion of S_3 follows a uniform distribution and thus has limited support. While it is well-known that such a distribution cannot be fitted exactly by a PH distribution because PH distributions have $f(t) > 0$ for all $t > 0$, in the case of S_3 our main focus is on the start of the distribution. This data set was obtained in a scenario where fitting the low quantiles (e.g. the 1% quantile) is important, while the higher quantiles may be ignored. We are thus especially interested in the lower part of the cumulative density function (CDF).

From Figure 5.4 we observe that G-FIT did not provide a good fit for the CDF of this distribution, as it does not fit the step at 0 at all. PhFit, on the other hand, was able to capture this feature well, and starts to diverge only at sample values above 6000. Simple k-means clustering gave slightly better results than G-FIT at the start of the distribution. Cluster refinement improved results significantly, irrespective of the refinement strategy. However, while clustering with refinement fitted the start of the distribution well, PhFit gave less oscillations at higher values. This may be due to the lower number of phases used by PhFit. Table 5.6 also shows that moment errors are comparable across all approaches, while log-likelihood values are much higher for clustering with refinement. Furthermore, the absolute error in the 1% quantile varies between approaches and is negligible when we use cluster refinement. Note that the density distance measures for clustering with refinement exhibit an interesting peculiarity, as these values are very high. This can be explained by the fact that in both cases one of the branches was a single exponential distribution with rate in the order of 10^{307}, resulting in a similar value for the density close to

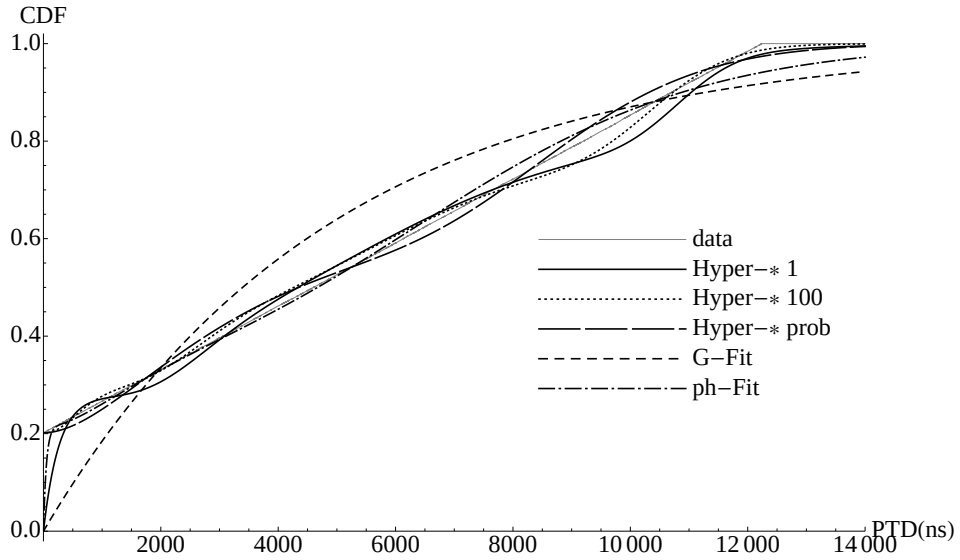


Figure 5.4: Empirical cumulative distribution function of the S_3 data set and CDFs of fitted distributions.

zero. The histogram used for computation of Δf, however, did not contain buckets with this height. This observation illustrates that under certain circumstances the density distance measure may be inappropriate for judging fitting quality.

5.4.2 Comparison of Cluster-based Fitting and Segmentation-based Fitting

We compare cluster-based fitting to our implementation of the segmentation-based approach SEM (see Section 5.2). Where available, we use the parameters found in [WLS08]. Wang et al. do not specify the initial values of the BEM algorithm, nor the number of iterations. With respect to initial value selection, we use the simple method proposed in [ESH03], which Wang et al. refer to. Also according to [ESH03], good results usually require as little as 5–10 iterations. Based on this statement, we set the number of iterations to 20.

We first compare the results obtained with our implementation to those published in [WLS08]. Like [WLS08], we use the NASA HTTP access log trace for July 1995 [ITA95].[1] From the trace different properties can be derived, e.g. the time between requests, time between requests from individual hosts, request sizes, etc.

[1]The Wireless trace also used in [WLS08] was not accessible anymore.

Method	n	Δf	ΔF	$\mathcal{L}$	e_1	e_2	e_3	e_q
k-Means	227	0.042	20.8	-908834.82	0.030	0.066	0.127	9.716
100 Iterations	132	4×10^{307}	13.7	1.352×10^7	2×10^{-16}	0.019	0.044	0.
Probabilistic Assignment	24	4×10^{307}	15.0	1.351×10^7	0.001	0.003	0.019	0.
G-FIT	20	0.055	73.8	-949118.80	7×10^{-15}	0.199	0.919	49.213
PhFit	15	0.0210	14.9	-885020.51	0.026	0.092	0.236	2.4134

Table 5.6: Quality measures for the S_3 data set.

k	1	2	3	4	5	
$r_{\text{Moments}}(k)$	1	0.97	0.93	0.87	0.68	
$r_{\text{Moments}}(k)$[WLS08]	1	0.98	0.98	1.0	0.97	
p	0.99	0.999	0.9999	0.99999	0.999999	0.9999999
$r_{\text{Quantile}}(p)$	0.89	0.89	0.79	0.34	0.32	0.15
$r_{\text{Quantile}}(p)$[WLS08]	0.90	0.93	1.15	0.89	1.15	0.92

Table 5.7: Comparison of quantitative measures for SEM implementations.

Although not explicitly specified by Wang et al., based on the mean and coefficient of variation it appears most likely that the data set used in [WLS08] consists of the non-zero request sizes, and thus we also use this data set.

Wang et al. evaluate fitting quality using the ratio between the fitted and empirical kth moment and the ratio between the p quantiles, defined as

$$r_{\text{Moments}}(k) := \frac{\mathrm{E}[\hat{X}^k]}{\mathrm{E}[X^k]} \quad \text{and} \quad r_{\text{Quantile}}(p) := \frac{\hat{F}^{-1}(p)}{F^{-1}(p)}, \tag{5.7}$$

respectively. Table 5.7 shows our results and those from [WLS08] for the first five moments and the higher quantiles starting at 0.99. Note that while we obtain similar values up to the fourth moment and the 99.99% quantile, with higher moments and larger quantiles our implementation of the algorithm gives worse results than those reported by Wang et al. This discrepancy may be a consequence of different choices for the parameter values that were not explicitly stated in [WLS08]. Nonetheless, the results are sufficiently similar to assume that our implementation reflects the behaviour of SEM.

We now consider fitting our data sets S_1, S_2, and S_3 using coefficient of variation thresholds $cv = 0.1$, 0.5, 0.6 and 1.01. The values 0.5 and 1.01 are used in [WLS08]; we use 0.1 and 0.6 to evaluate the sensitivity of the algorithm to the threshold value.

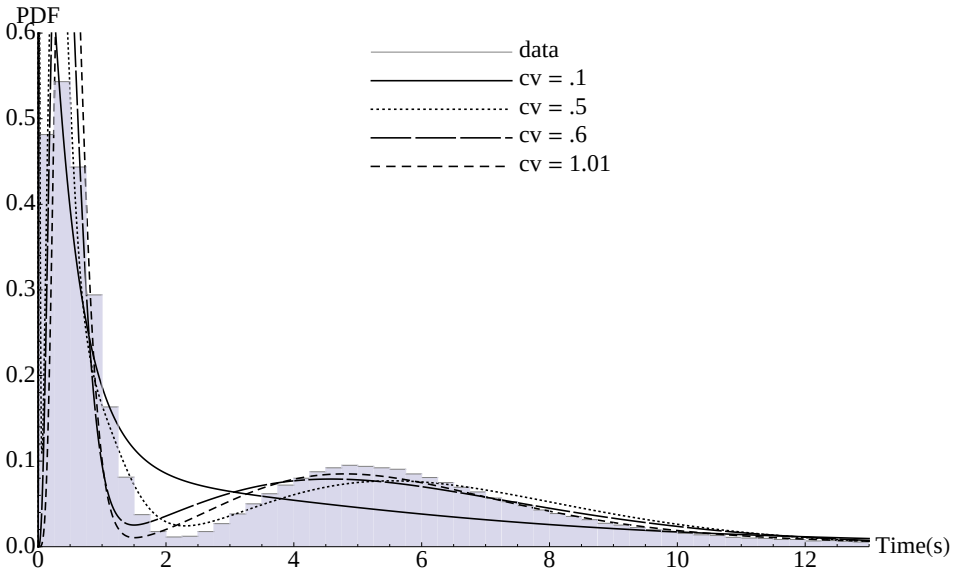


Figure 5.5: SEM fitting for S_1 with different thresholds cv for the coefficient of variation.

Figure 5.5 shows the histogram of S_1 and the densities returned by the segmentation approach. For a threshold of 0.5 the segmentation-based approach fits the density quite well, while for other thresholds the fit is worse; in particular, the dent in the density is not captured well. According to the quantitative results in Table 5.4, the SEM algorithm performs worse than either of the other tools, even when the size of the fitted distribution is larger.

As discussed before, the S_2 data is more difficult to fit than S_1. From visual inspection of the fitting results (Figure 5.6) we observe that, irrespective of the cv threshold, SEM failed to fit the important properties of the data set. While difficult to see in the plot, low cv values helped to capture the first peak. The remaining peaks are not represented with any threshold. Furthermore, for $cv = 1.01$ only a single segment was found, which does not reflect any of the features of the density. This data set illustrates that SEM fails to fit peaks in the right part of the density. The quantitative measures (Table 5.6) underline this observation. However, note that SEM may fit the first three moments quite well, especially with a cv threshold of 0.1.

In our experiments the SEM algorithm did not return feasible results for the S_3 data set. The reason for this is that the implementation of the EM algorithm in BEM only returns valid values for samples that are strictly larger than zero. We

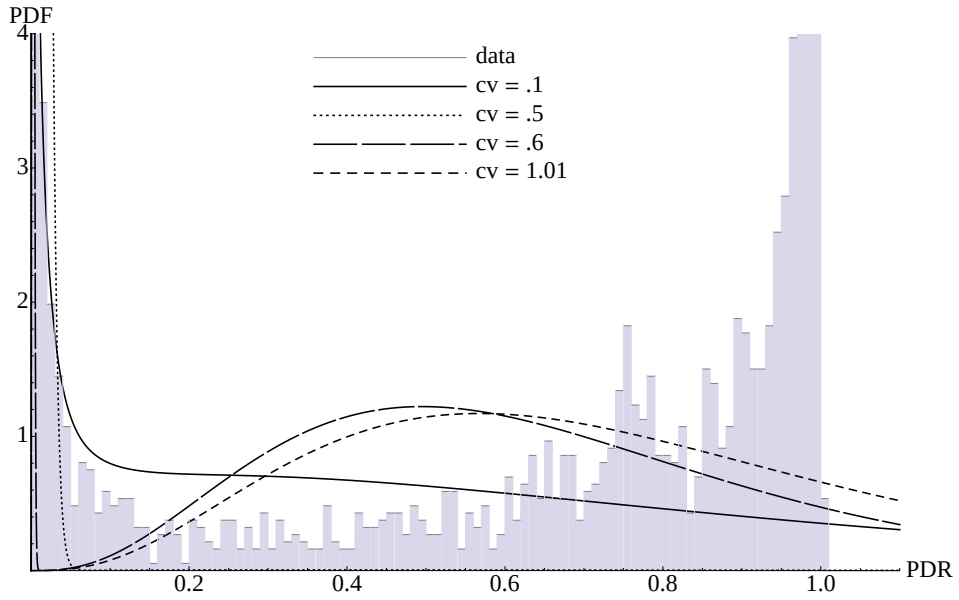


Figure 5.6: SEM fitting for S_2 with different thresholds cv for the coefficient of variation.

tried to fit this data set by manually removing the zeros and fitting the remaining uniform distribution. However, as we could not obtain good fitting results for the uniform distribution either, we omit evaluation using the S_3 data set.

As expected, the segmentation approach is sensitive to the threshold: Low threshold values result in many small segments that can capture many oscillations, while large values give fewer segments. If the threshold value is chosen appropriately, the algorithm may provide a good fit of the density; however, the optimal choice of threshold depends on the distribution. The bad fit for peaks in the right part of the density can be explained from the observation that the segmentation algorithm tends to produce more and smaller segments for lower sample values than for higher sample values. This behaviour follows directly from the use of the coefficient of variation as a measure for the size of the segment: With low values, segments have low means, and thus only small variation can be tolerated before the threshold is reached. With segments containing higher values, means are larger, and the threshold is only reached for large variation. While this might be addressed to some extent by choosing a smaller threshold, a smaller threshold also increases the number of peaks in the fitted density at the beginning of the data set, and may significantly decrease fitting quality, as with S_1.

We add that we also considered a combination of the segmentation-based algorithm and G-FIT for fitting of the partitions. With this implementation we run

G-FIT on the partitions obtained by the segmentation algorithm. We specify a maximum number of phases and let G-FIT automatically select the optimal number of branches and the optimal assignment of phases to branches. While the results for the NASA trace differed, in the evaluation with our data sets this implementation gave very similar results. Since G-FIT employs a different variant of the EM algorithm, this observation indicates that the low fitting quality of SEM is a result of segmentation rather than of the fitting of the segments.

5.4.3 Discussion

Our evaluation results are two-fold. First, we observe that our clustering-based fitting approach performs well for the data sets considered here, as it is especially suited for fitting several sharp peaks of the density. Second, we note that for these data sets results from our approach are generally much better than those of the segmentation-based approach by [WLS08], and that our approach may also capture the empirical density much better than G-FIT or PhFit. Both tools, on the other hand, typically give distributions with lower moment errors. Third, the good fitting quality of the clustering-based approach is at least partly due to the fact that the approach may use long branches, and thus give large distributions. On the other hand, the probabilistic assignment strategy may reduce the number of phases without decreasing fitting quality.

5.5 Case-Study: Application of HyperStar in Algorithm Development

We have already pointed out that one of the design goals of HyperStar was to enable users who are not experts in the area of PH distributions to obtain good fits to empirical data. This particular use is greatly facilitated by the fact that HyperStar uses a clustering algorithm which can be parameterised in an intuitive way by marking important sections of the histogram. In this section we illustrate the advantages of using HyperStar as a graphical user interface to prototype implementations of new fitting approaches. We do so by providing a case-study where we use HyperStar to evaluate a new approach for general PH fitting. We first introduce the `MonoFit` algorithm for fitting general PH distributions in FE-diagonal form to data sets. We then show the evaluation of a prototype implementation of the algorithm using HyperStar.

5.5.1 The MonoFit Algorithm for Fitting FE-Diagonal Representations to Data

Most approaches for fitting phase-type distributions focus on the APH sub-class, or even sub-classes thereof, e.g. the Hyper-Erlang distributions, because with the reduced number of free parameters required by these sub-classes fitting efficiency and fitting quality can be improved (see e.g. [ANO96, LA96, HT02, TBT06, Dan10]). One of the few tools that support the whole PH class is EMpht [ANO96], but even the authors of the tool suggest limiting the structure of the resulting distribution to improve fitting results. Such limitations can imply limitations to the class of the distributions that are fitted.

We propose the use of the FE-diagonal form (see Chapter 2) in general PH fitting. Due to the special structure as a chain of Feedback-Erlang blocks, FE-diagonal forms require a lower number of parameters than general representations. Furthermore, the FE-diagonal form is a generalisation of the Monocyclic form and can therefore express general PH distributions. The algorithm we develop in the following is an application of the well-known Nelder/Mead algorithm for direct optimisation [NM65, JS03, SN09]. The Nelder/Mead algorithm explores the surface of a function (e.g. the density distance) by constructing a series of simplices (i.e. polytopes with $n + 1$ vertices in an n-dimensional space). Starting with random initial vertices, the simplex moves towards lower values and avoids higher values of the distance function by changing its size and shape accordingly, finally contracting around a local optimum. While it is known that the algorithm may converge slowly, or fail to converge to an optimum at all [JS03, SN09], it has the advantage that it does not require the computation of derivatives of the distance function and thus involves only a small number of function evaluations [NM65, SN09].

The Nelder/Mead algorithm has been considered for PH fitting before in [Fad98, IH05, MR93]. These approaches are limited with respect to the class they support. Both Faddy [Fad98] and Isensee and Horton [IH05] consider the APH class in one of the canonical forms proposed in [Cum82]. Faddy [Fad98] uses the canonical form CF-A, where the first state is the initial state, and all states can be entered from this state (cf. [Cum82]). Iseensee and Horton [IH05] employ the CF-1 form, described in Section 2.4. Malhotra and Reibman [MR93] propose the use of the Nelder/Mead algorithm to fit Erlang and Hyper-Erlang distributions to analytical distributions, e.g. the Lognormal distribution. Observations by these authors indicate that the Nelder/Mead algorithm may perform quite well for fitting sub-classes of PH distributions to data sets or analytical distributions. The MonoFit algorithm, described in the following, applies the Nelder/Mead approach for fitting general PH distributions in FE-diagonal form to data sets.

5.5.1.1 The Fitting Problem for FE-Diagonal Representations

We first formalise the fitting problem for FE-diagonal representations. We start by considering that, in addition to providing a good fit, we also want the algorithm to be user-friendly. We therefore we want the algorithm to require only a minimum of parameters to be specified by the user. The requirement that the representation be FE-diagonal already imposes constraints on the structure of the sub-generator matrix. This leaves three parameters to be specified: The size of the distribution, the number of Feedback-Erlang blocks, and the lengths of blocks, i.e. the allocation of phases to blocks. Each vertex of the simplex is then expressed by the following parameter vector

$$\boldsymbol{\theta} = (\boldsymbol{\alpha}, ((b_1, \lambda_1, z_1), \ldots, (b_m, \lambda_m, z_m)) \in \mathbb{R}^{n+3m}. \tag{5.8}$$

The Nelder/Mead algorithm searches the $(n + 3m)$-dimensional space of parameter vectors $\boldsymbol{\theta}$ for an optimum. One implicit prerequisite for the algorithm is that all vertices are in the same space, i.e. that all parameter vectors have the same length. Note that if we ensure that the overall size of the distribution,

$$n = \sum_{i=1}^{m} b_i \tag{5.9}$$

and the number of blocks, m, stay constant, we may permit changes to the lengths of the blocks $b_1, \ldots, b_m$ without changing the length of the parameter vector $\boldsymbol{\theta}$. Consequently, the search for an optimal allocation of phases to FE blocks can be incorporated in the optimisation process.

Then, two parameters are left: The number of phases, n, and the number of blocks, m. We cannot apply the previous argument to eliminate the necessity of specification of these parameters, since any adjustments to them change the length of the vertices. Repeated execution of a given fitting algorithm with different values for these parameters and selection of the best fit (as implemented in G-FIT) may help to find a better fitting in cases where the required representation size or number of blocks are not known.

Note that the number of phases and number of blocks are useful in choosing a distribution suitable for efficient random-variate generation. In particular, the number of blocks immediately translates to the maximal number of logarithm operations that may be required, and thus gives a measure for worst-case costs.

The fitting problem for FE-diagonal representations can now be stated as follows: Let $\bar{n} = n + 3m$ be the number of parameters of an FE-diagonal representation. Given the parameter vector

$$\boldsymbol{\theta} = (\boldsymbol{\alpha}, ((b_1, \lambda_1, z_1), \ldots, (b_m, \lambda_m, z_m)) \in \mathbb{R}^{\bar{n}}, \tag{5.10}$$

minimise a distance function $\mathcal{D}(F_{\boldsymbol{\theta}}, F)$ between the approximating distribution $F_{\boldsymbol{\theta}}$ defined by $\boldsymbol{\theta}$ and the empirical distribution F, subject to the following constraints:

$$\begin{aligned} \boldsymbol{\alpha}\mathbb{1} &= 1, && (5.11)\\ \alpha_i &\geq 0, && (5.12)\\ \lambda_j &> 0, && (5.13)\\ z_j &\geq 0, && (5.14)\\ z_j &< 1, && (5.15)\\ b_j &\in \mathbb{N}\backslash\{0\}, && (5.16)\\ \sum_{l=1}^{m} b_l &= n, && (5.17) \end{aligned}$$

for $i = 1, \ldots, n$ and $j = 1, \ldots, m$. Constraints (5.11) through (5.15) ensure that the result is a phase-type distribution. Equation (5.16) and (5.17) guarantee that the number of blocks and the size of the distribution stay constant. We note in passing that (5.16) and (5.17) imply that the feasible region is not convex, and thus the Frank/Wolfe method used in PhFit cannot be applied to this problem.

The algorithm may be extended to fit only Monocyclic representations by adding the following constraint, which enforces FE-block ordering according to the absolute value of the dominant eigenvalue:

$$\lambda_i \left(1 - z_i^{\frac{1}{b_i}}\right) \leq \lambda_{i+1} \left(1 - z_{i+1}^{\frac{1}{b_{i+1}}}\right). \quad (5.18)$$

However, as our initial experiments indicated that this additional constraint did not improve fitting results we do not use it in the following.

5.5.1.2 Application of the Nelder/Mead Algorithm in FE-diagonal Fitting

The original Nelder/Mead algorithm is only suited for unconstrained optimisation, but there exist several approaches that extend it to support constraints. In constrained optimisation, both penalties and projection are applied. Penalties assign a large cost value to points outside the feasible region [JS03], and have been proposed in the original paper [NM65]. Projection moves points outside the feasible region into the interior of the region or onto its borders, but may lead to linearly dependent vertices (a 'collapse' of the simplex into fewer than n dimensions), which may result in convergence to a bad local optimum [Box65, LLRG03].

Both extensions have disadvantages when the algorithm is applied in PH fitting. First, note that constraints (5.16) and (5.17) require that the sizes of all blocks are

natural numbers larger than zero and that the size of the distribution stays constant. Both constraints are very likely to be violated by a new parameter vector constructed in either of the steps. Consequently, assigning a penalty to these constraints results in many rejections of parameter vectors. We therefore apply the projection approach to move points that violate (5.16) and (5.17) back onto the boundaries of the feasible region. We also use projection to ensure (5.11), (5.12) and (5.14). Appendix D gives details on the projection methods used.

Constraints (5.13) and (5.15) are difficult to ensure by projection, since the boundary is only defined by a strict inequality. We therefore apply penalties to enforce these constraints.

We note that in Monocyclic fitting (using the additional constraint (5.18)), projection would be an expensive method for ensuring the eigenvalues ordering, because it requires re-ordering the Feedback-Erlang chain and computing a new initial vector. A penalty may thus be more appropriate for ensuring the eigenvalues ordering.

Pseudo-code for our Nelder/Mead-based fitting algorithm for FE-diagonal representations is shown in the following:

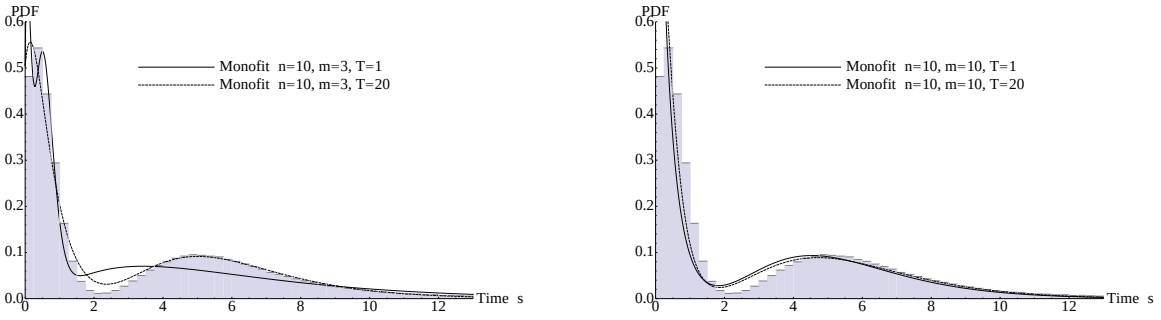
Procedure `MonoFit`$(\mathcal{D})$:
Create $N * (\bar{n}+1)$ random vertices $V := \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{N(\bar{n}+1)}\}$.
Sort V such that $\mathcal{D}(F_{\boldsymbol{\theta}_i}, F) \leq \mathcal{D}(F_{\boldsymbol{\theta}_{i+1}}, F)$ for $i = 1, \ldots, N(\bar{n}+1)$ and select the first $\bar{n}+1$ vertices: $V := \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{\bar{n}+1}\}$.
$R := 1$
while no convergence $\wedge$ $R < R_{\max}$ **do**
  $\boldsymbol{\theta}_c := \frac{1}{\bar{n}} \sum_{i=1}^{\bar{n}} \boldsymbol{\theta}_i$
  $\boldsymbol{\theta}_r := \boldsymbol{\theta}_c + w_1(\boldsymbol{\theta}_c - \boldsymbol{\theta}_{\bar{n}+1})$
  if $\mathcal{D}(F_{\boldsymbol{\theta}_1}, F) \leq \mathcal{D}(F_{\boldsymbol{\theta}_r}, F) \leq \mathcal{D}(F_{\boldsymbol{\theta}_{\bar{n}}}, F)$ **then**
    $\boldsymbol{\theta}_{\bar{n}+1} := \boldsymbol{\theta}_r$
    Sort V.
  else
    if $\mathcal{D}(F_{\boldsymbol{\theta}_r}, F) < \mathcal{D}(F_{\boldsymbol{\theta}_1}, F)$ **then**
      $\boldsymbol{\theta}_e := \boldsymbol{\theta}_c + w_2(\boldsymbol{\theta}_r - \boldsymbol{\theta}_c)$
      if $\mathcal{D}(F_{\boldsymbol{\theta}_e}, F) < \mathcal{D}(F_{\boldsymbol{\theta}_r}, F)$ **then**
        $\boldsymbol{\theta}_r := \boldsymbol{\theta}_e$
      end if
      $\boldsymbol{\theta}_{\bar{n}+1} := \boldsymbol{\theta}_r$
      Sort V.
    else
      if $\mathcal{D}(F_{\boldsymbol{\theta}_r}, F) > \mathcal{D}(F_{\boldsymbol{\theta}_{\bar{n}}}, F)$ **then**
        $$\boldsymbol{\theta}_s := \begin{cases} \boldsymbol{\theta}_c + w_3(\boldsymbol{\theta}_r - \boldsymbol{\theta}_c) & \mathcal{D}(F_{\boldsymbol{\theta}_r}, F) < \mathcal{D}(F_{\boldsymbol{\theta}_{\bar{n}+1}}, F) \\ \boldsymbol{\theta}_c + w_3(\boldsymbol{\theta}_{\bar{n}+1} - \boldsymbol{\theta}_c) & \mathcal{D}(F_{\boldsymbol{\theta}_r}, F) \geq \mathcal{D}(F_{\boldsymbol{\theta}_{\bar{n}+1}}, F) \end{cases}.$$
        if $\mathcal{D}(F_{\boldsymbol{\theta}_s}, F) < \min(\mathcal{D}(F_{\boldsymbol{\theta}_r}, F), \mathcal{D}(F_{\boldsymbol{\theta}_{\bar{n}+1}}, F))$ **then**

```
            θ_{n̄+1} := θ_s
          else
            θ_i := ½(θ_1 + θ_i) for i = 2, ..., n̄ + 1
          end if
          Sort V.
        end if
      end if
    end if
    R := R + 1
  end while
  return θ_1.
```

The algorithm first generates a large set of random, but feasible vertices, from which the $\bar{n}+1$ vertices with the lowest distance function values are selected to form the initial simplex. In order to reduce computational cost, we do not explicitly check whether the vertices are linearly independent. While linearly dependent vertices (i.e. a collapse of the Simplex [SN09]) might result in bad convergence behaviour and bad fitting [SN09], this case appears unlikely with randomly selected vertices. The algorithm moves and modifies the simplex according to the Nelder/Mead algorithm (we follow [SN09] throughout the following discussion): The list of vertices is always kept ordered such that $\mathcal{D}(\boldsymbol{\theta}_1) \leq \ldots \mathcal{D}(\boldsymbol{\theta}_{\bar{n}})$.[2] Furthermore, throughout the algorithm vertices that do not represent a Markovian FE-diagonal representation of a phase-type distribution and are thus located outside the feasible region are projected onto the borders, if possible, or penalised if projection is not possible for this constraint (see above).

The vertex $\boldsymbol{\theta}_c$ is the center of the 'best side', which is formed by the $\bar{n}$ best vertices. $\boldsymbol{\theta}_r$ is the point that results from reflecting the worst point ($\boldsymbol{\theta}_{\bar{n}+1}$) through $\boldsymbol{\theta}_c$. If the new vertex $\boldsymbol{\theta}_r$ is lower than the current worst point (i.e. it has a lower distance function value), but not better than the current best vertex, $\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_r$ replaces the current worst point and the algorithm enters the next iteration. Graphically, this can be seen as the simplex being *reflected* along its best side. If $\boldsymbol{\theta}_r$ is better than the current best vertex, $\boldsymbol{\theta}_1$, the algorithm checks if further improvement is possible in this direction, by computing an additional point $\boldsymbol{\theta}_e$ that lies on the line through $\boldsymbol{\theta}_c$. If $\boldsymbol{\theta}_e$ is better than $\boldsymbol{\theta}_r$, the worst point is replaced by $\boldsymbol{\theta}_e$, which amounts to an *expansion* of the simplex towards a better area. If $\boldsymbol{\theta}_e$ is worse than $\boldsymbol{\theta}_r$, the simplex is only reflected. In both cases, a new iteration starts after the modification of the simplex. If $\boldsymbol{\theta}_r$ is worse than the current second-worst vertex, $\boldsymbol{\theta}_{\bar{n}}$, the simplex *contracts* to avoid this point. Contraction occurs by computing a new point $\boldsymbol{\theta}_s$. If $\boldsymbol{\theta}_r$ is still better than the worst point, $\boldsymbol{\theta}_s$ is between $\boldsymbol{\theta}_c$ and $\boldsymbol{\theta}_r$ (i.e. outside the simplex),

[2] Note that, although the description of the algorithm given above uses explicit `Sort` operations, our implementation applies insertion into a sorted list and avoids explicit sorting, wherever possible (cf. [SN09]).



(a) $m = 3$ (b) $m = 10$

Figure 5.7: MonoFit fitting results for the S_1 data set with $n = 10$.

otherwise, $\boldsymbol{\theta}_s$ lies between $\boldsymbol{\theta}_c$ and $\boldsymbol{\theta}_{\bar{n}+1}$ (inside the simplex). $\boldsymbol{\theta}_s$ replaces the worst point only if it entails an improvement, otherwise, the whole simplex *shrinks* by moving all vertices close to the current best point, $\boldsymbol{\theta}_1$.

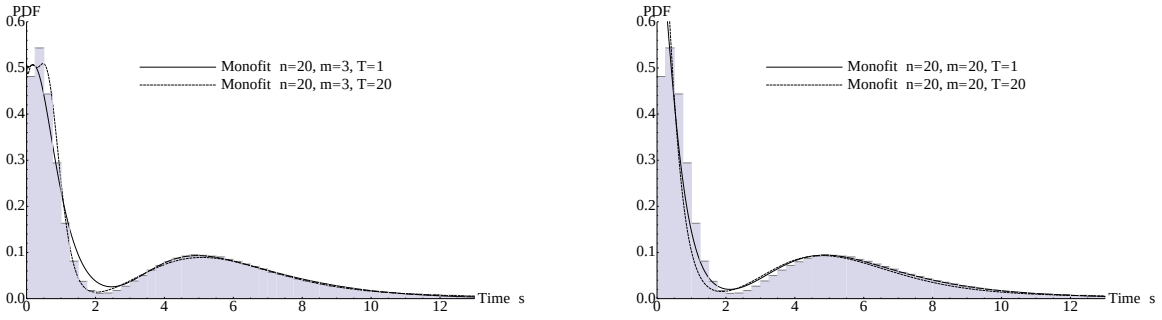
The algorithm terminates if the vertices are sufficiently similar (according to the variance criterion proposed by [SN09]) or if a maximum number of iterations $R_{\max}$ has been exceeded. The vertex with the lowest value for the distance function is returned. As the Nelder/Mead algorithm is sensitive to variations of the initial parameters [SN09], the above procedure is encapsulated in a routine that allows us to run it multiple times with different random seeds and selects the best fit. This also helps to counteract convergence to bad local optima.

5.5.2 MonoFit Evaluation Using HyperStar

We evaluate a prototype implementation of MonoFit. The prototype is written in Mathematica. With the Mathematica interface of HyperStar, we pass the histogram of the data set and the weighting function as `Piecewise[]` functions into Mathematica and call our Mathematica function `MonoFit` that implements the algorithm. Our implementation then returns the fitted distribution as an $(\boldsymbol{\alpha}, \mathbf{Q})$ tuple.

We use both the S_1 and S_2 data sets in our evaluation. For the S_1 data set we know that it can be fitted well by an APH distribution with 20 phases. For the S_2 data set our results in Section 5.4 indicate that it can be fitted well by Hyper-Erlang distributions, but is in general difficult to fit by a PH distribution. We omit the S_3 data set because our MonoFit prototype does not support fitting this distribution.

We start by a discussion of the fitting results using MonoFit. MonoFit has three important parameters: The number of phases n, the number of FE blocks m and the number of repeated invocations of the algorithm, T. In our evaluation we use two different settings for each parameter, as follows: We consider $n = 10$ and $n = 20$, setting $m = 3$ and $m = n$ and the number of trials to $T = 1$ or $T = 20$, resulting



(a) $m = 3$ (b) $m = 20$

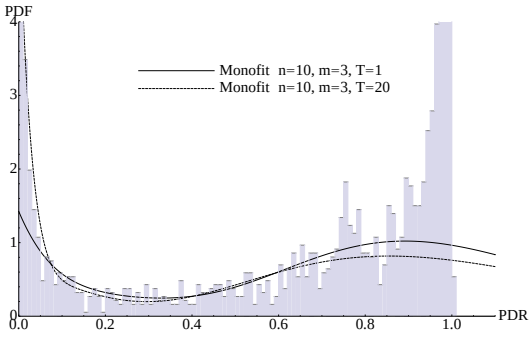
Figure 5.8: MonoFit fitting results for the S_1 data set with $n = 20$.

n	m	T	Δf	ΔF	$\mathcal{L}$	e_1	e_2	e_3
10	3	1	14.1865	16.5462	-19673.145	0.036787	0.0250188	0.138162
10	3	20	7.20281	14.9384	-19404.429	0.069995	0.1996175	0.356007
10	10	1	20.6257	30.4309	-20470.566	0.172711	0.3153657	0.475116
10	10	20	14.4712	24.9562	-19897.344	0.141650	0.2748038	0.434301
20	3	1	6.33659	16.5362	-19435.324	0.080979	0.2189639	0.380749
20	3	20	4.63531	12.4760	-19121.035	0.069347	0.1459737	0.231443
20	20	1	10.5135	14.4013	-19452.872	0.080661	0.1718633	0.292308
20	20	20	14.0634	11.9797	-19747.157	0.003726	0.2177268	0.848227

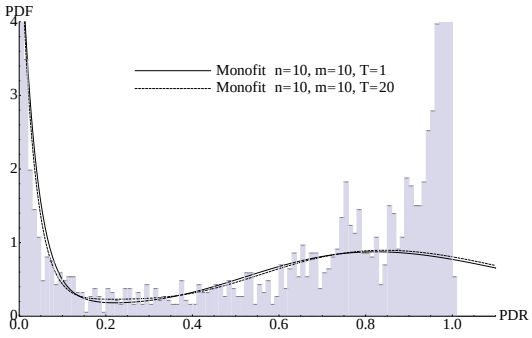
Table 5.8: Quality measures for MonoFit with the S_1 data set.

in 8 different parameter settings. All runs start with the same random seed of 1, which is set before the first trial of the run. This ensures that evaluation results are repeatable and that the results are only influenced by the choice of parameters. The other paramaters of the algorithm are set as follows: The number of initial simplices is $N = 1000$, the maximal number of iterations is $R = 10000$, the convergence threshold is 10^{-9}, and the Nelder/Mead parameters are $w_1 = 1, w_2 = 2, w_3 = 0.5$.

Figures 5.7 and 5.8 show the fitting results for S_1 with $n = 10$ and $n = 20$, respectively. The figures again display the histogram of the data set and the densities of the fitted distributions. In Figure 5.7(a) we observe that increasing the number of trials from 1 to 20 improves the fitting of the density if the distribution consists of $m = 3$ FE blocks. Figure 5.7(b) shows results for $m = 10$, i.e. an APH distribution in bi-diagonal form. In this case, the fit of the density is worse than for $m = 3$ and does not improve with an increased number of trials. This can also be observed from the quantitative measures, shown in Table 5.8. In Figure 5.8 we increased the number of phases to $n = 20$. With $m = 3$ FE blocks and 20 trials (Figure 5.8(a)) the

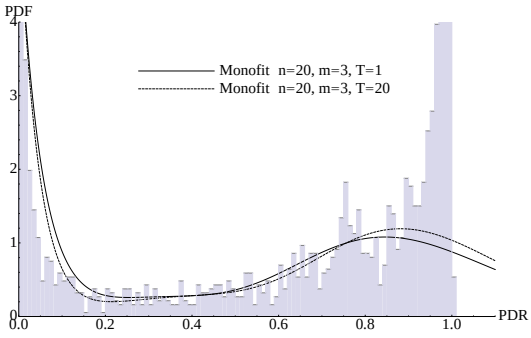


(a) $m = 3$

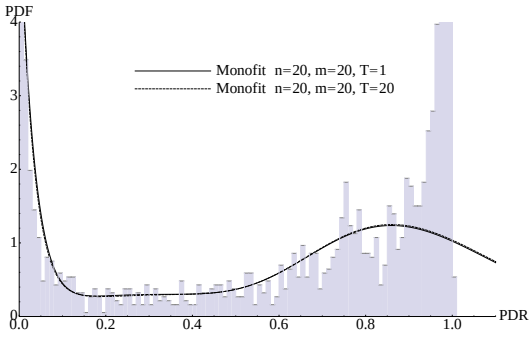


(b) $m = 10$

Figure 5.9: MonoFit fitting results for the S_2 data set with $n = 10$.



(a) $m = 3$



(b) $m = 20$

Figure 5.10: MonoFit fitting results for the S_2 data set with $n = 20$.

algorithm yields a very good fit of the density, whereas with only 1 trial the valley in the density is not captured well. By increasing the number of blocks to $m = 20$ (resulting in an APH with 20 phases in bi-diagonal form), we obtain a fitting of the density that is better than for $n = 10$, but worse than that for $n = 20, m = 3$. In this case, increasing the number of trials actually decreases the fitting quality slightly, which may be due to the discrete computation of the density distance measure in the algorithm. The quantitative results again corroborate the observations of our visual inspection. For this data set, $n = 20, m = 3, T = 20$ appears to result in the best fit, although most configurations gave acceptable results. Note that this configuration requires only 9 parameters for the sub-generator matrix, compared to 20 with an APH distribution.

In Figures 5.9 and 5.10 we show the fitted densities for the S_2 data set with $n = 10$ and $n = 20$ phases. None of the fitted distributions represent the two distinct peaks around 0.8 and 1 well. The distributions differ with respect to the first peak around

n	m	T	Δf	ΔF	$\mathcal{L}$	e_1	e_2	e_3
10	3	1	501.92852	136.6637	-296.456	0.41320	0.841243	1.58995
10	3	20	482.71067	68.36005	-122.538	0.32265	0.811984	1.72801
10	10	1	500.47813	39.14751	-16.8046	0.21463	0.537903	1.11993
10	10	20	482.21270	54.16396	-11.6291	0.24505	0.584384	1.18636
20	3	1	520.49516	34.00178	208.5858	0.02816	0.1279956	0.30473
20	3	20	478.35617	19.75485	246.1847	0.10031	0.234295	0.44290
20	20	1	453.65508	34.34955	281.5916	0.11835	0.229114	0.40170
20	20	20	449.36469	41.15216	279.9134	0.13145	0.246642	0.42344

Table 5.9: Quality measures for MonoFit with the S_2 data set.

0 and the low plateau between 0.1 and 0.7. With 10 phases and 3 FE blocks the result of fitting with only one trial fails to capture the peak at the beginning of the density completely, but reflects the plateau rather well (Figure 5.9(a)). With 20 trials, the first peak is represented very well, but the height of the peaks at the end is lower than with 1 trial. For fitting 10 FE blocks instead of 3 (Figure 5.9(b)) the fitting quality is not influenced by the number of trials. With 20 phases (Figure 5.10) we obtain a better fit of the plateau, and all configurations represent the first peak well. With 20 phases, the results with $m = 20$ blocks appear to capture the general shape of the density best. This is also visible in the quantitative results, shown in Table 5.9.

5.5.3 Comparison to EMpht

We compare the results of MonoFit to those of the EMpht tool by Asmussen et al. [ANO96]. In contrast to MonoFit, EMpht uses the expectation-maximisation algorithm. EMpht is the only tool available that uses general phase-type distributions. EMpht is particularly interesting for our comparison because it can be parameterised to only fit PH representations with a specified structure. The user can specify which entries of the resulting initial vector and generator matrix shall be equal to zero. We use this option to define a structure that is similar to a chain of Feedback-Erlang blocks, as follows: All entries of the initial vector are allowed to be non-zero. In each row of the sub-generator matrix, the elements of the diagonal and the upper diagonal are non-zero and equal. The only other elements in each row that can be non-zero are at the feedback points. For instance, for a single block

of size b with one feedback we define the following matrix:

$$\mathbf{Q} = \begin{pmatrix} -\lambda_1 & \lambda_1 & & & \\ & -\lambda_2 & \lambda_2 & & \\ & & \ddots & \ddots & \\ & & & -\lambda_{(b-1)} & \lambda_{(b-1)} \\ \lambda_{\mathrm{FB}} & & & & -\lambda_b \end{pmatrix}. \tag{5.19}$$

Note that we cannot specify equality of all rates in a block, i.e. $\lambda_{11} = \lambda_{22} = \cdots = \lambda_{bb}$ cannot be guaranteed by the algorithm. The results of EMpht thus resemble a chain of FE blocks, but do not share the sparse representation available for chains of FE blocks, since in each block all rates have to be specified explicitly. In order to define the block shown in (5.19), the length b, the feedback rate λ_{FB} and the b rates on the diagonal have to be specified. A single block thus requires $b + 2$ parameters. A chain of m blocks can be defined by the following tuple:

$$\left(\left(b_1, \lambda_1^{(1)}, \ldots, \lambda_{b_1}^{(1)}, \lambda_{\mathrm{FB}}^{(1)}\right), \ldots, \left(b_m, \lambda_1^{(m)}, \ldots, \lambda_{b_m}^{(m)}, \lambda_{\mathrm{FB}}^{(m)}\right)\right), \tag{5.20}$$

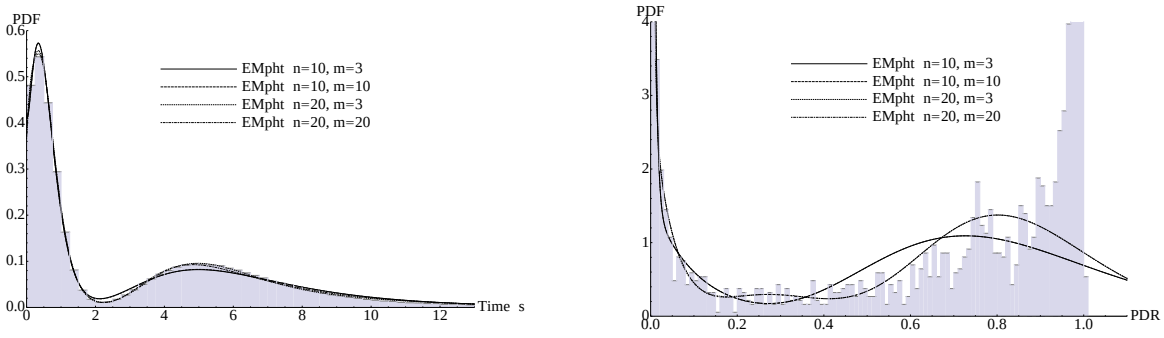
where $b_1, \ldots, b_m$ give the lengths of blocks, $\lambda_1^{(i)}, \ldots, \lambda_{b_m}^{(i)}$ are the rates on the diagonal, and $\lambda_{\mathrm{FB}}^{(i)}$ is the feedback rate of the ith block. The definition of a chain of such blocks therefore requires $2m + \sum_{i=1}^{m} b_i$ parameters. In contrast, an FE block requires specification of the length, rate, and feedback probability, and can therefore be given by 3 parameters. The number of parameters for specifying the structure of an FE-diagonal sub-generator matrix with m blocks is then

$$3m \leq 2m + \sum_{i=1}^{m} b_i, \tag{5.21}$$

where the left and right side are equal for $b_1 = \cdots = b_m = 1$.

The fact that all rates have to be specified explicitly also implies that the results of EMpht cannot be used in efficient random-variate generation using the `FE-diagonal` method.

We use the size of the distribution and the structure of the sub-generator matrix to reflect the different parameter combinations chosen for MonoFit with EMpht. We study distributions of size $n = 10$ and $n = 20$. For each size, we set the number of blocks to $m = 3$ and $m = n$. Since EMpht does not support automatic block size selection, we specify the sizes of blocks explicitly, by defining an appropriate matrix. For $n = 10$ and $m = 3$ the matrix consists of 3 blocks of size 1, 3, and 6 along the diagonal (in this order). For $n = 20$ and $m = 3$, the blocks have sizes 1, 3, and 16. For $n = m$ block sizes are 1. Although other block sizes might have been chosen,



(a) S_1 (b) S_2

Figure 5.11: EMpht fitting results for the S_1 and S_2 data sets.

n	m	Δf	ΔF	$\mathcal{L}$	e_1	e_2	e_3
10	3	6.15625	6.31308	-19114.38	5.771×10^{-7}	0.01222	0.04553
10	10	6.02399	6.17034	-19109.85	2.321×10^{-7}	0.01278	0.04661
20	3	3.89285	2.47231	-19049.26	9.798×10^{-8}	0.00455	0.00517
20	20	4.40730	3.60805	-19060.39	7.678×10^{-7}	0.01054	0.02947

Table 5.10: Quality measures for EMpht with the S_1 data set.

n	m	Δf	ΔF	$\mathcal{L}$	e_1	e_2	e_3
10	3	502.6733	39.33616	271.3386	3.273×10^{-7}	0.05702	0.19081
10	10	502.7624	39.33086	271.3038	7.192×10^{-7}	0.05702	0.19079
20	3	458.0315	23.52119	467.2090	1.519×10^{-7}	0.02257	0.07478
20	20	458.4278	23.55124	466.2455	1.655×10^{-7}	0.02266	0.07504

Table 5.11: Quality measures for EMpht with the S_2 data set.

these already provided sufficiently good results. We set the random seed to 42 and the number of iterations in the EM algorithm to 500. All other parameters were left at their default values. We consider only one trial for each configuration, because in contrast to MonoFit EMpht requires manual configuration of individual trials, and because even with one trial we obtained good results.

Figure 5.11 shows histograms and fitted densities returned by the EMpht tool for both data sets. For the S_1 data set (Figure 5.11(a)), we observe that both $n = 10$ and $n = 20$ provide a good fit, and that with $n = 20$ the fitted distributions capture the empirical density perfectly. The number of blocks does not appear to have any effect on the fitting quality. Similar observations can be made for S_2 (Figure 5.11(b)):

Again, with a higher number of phases the fitting quality improves. In this case, 20 phases enabled a good fit of the first peak and the plateau, and an approximation for the second peaks. Note that in Figure 5.11(b) the densities overlap for $n = 10, m = 3$ and $n = m = 10$, and the same holds for $n = 20$. Closer investigation of the fitted distributions shows that for this data set EMpht produced almost identical representations irrespective of the number of blocks, i.e. the feedback rate was zero even for the configurations with $m = 3$, or, equivalently, EMpht always fitted an APH distribution. As with MonoFit, none of the configurations captured the two peaks at the end as distinctive features. The visual results are corroborated by the quantitative measures shown in Table 5.11.

5.5.4 Discussion

In our evaluation of MonoFit using HyperStar as a graphical user-interface we observe that MonoFit is capable of providing good fitting results for the S_1 and S_2 data sets. Comparing the results of EMpht to those of MonoFit we note that EMpht's fitting quality is equal or better than that of MonoFit. In particular, EMpht consistently provides higher likelihood values and lower moment errors.

On the other hand, the distributions returned by MonoFit tend to require less parameters, and enable efficient random-variate generation using the `FE-diagonal` method described in Chapter 3. EMpht's results do not lend themselves to sparse representations. With $n = 10, m = 3$, EMpht requires $6+(1+3+6) = 16$ parameters for the sub-generator matrix (compared to $3 \cdot 3 = 9$ parameters for the FE-diagonal representation given by MonoFit). With $n = 20, m = 3$, the number of parameters for the matrix rises to $6 + 1 + 3 + 16 = 26$, while an FE-diagonal matrix can still be given by 9 parameters. For $n = m$, on the other hand, both EMpht and MonoFit require the same number of parameters to store the representation.

We can thus summarise that the MonoFit algorithm is especially suited for applications where sparse representations or FE-diagonal representations are required. However, improvements should still be explored. For instance, different distance measures might improve fitting results,

5.6 Concluding Remarks

This chapter discussed methods and tools for fitting phase-type distributions to data. We presented an overview of existing fitting methods, taking into account both differences in the underlying approach and in usability.

We described a cluster-based fitting approach for phase-type distributions and showed that the algorithm performs well for identifying and fitting characteristics of the density of typical empirical distributions. Our algorithm is particularly suited to fitting sharp peaks of the density and performs better than algorithms that use

a fixed threshold for partitioning of the data. We discussed our implementation of the algorithm in the HyperStar tool. HyperStar enables an intuitive approach to phase-type fitting and can be easily extended by new fitting methods.

In a case-study we illustrated the use of HyperStar in prototyping new fitting algorithms. We developed an algorithm for fitting FE-diagonal representations to data. Using HyperStar, we then evaluated the algorithm. The algorithm shows promising results with little user interaction and returns compact representations. In comparing the algorithm to the EMpht tool we observed that the latter may return better fittings, but at the cost of less compact representations.

Part III

Applications

Chapter Six

On Stochastic Fault-Injection for IP-Packet Loss Emulation

In Part I we discussed stochastic fault models for system evaluation and provided a survey of existing fault models. In this chapter we focus on the network and illustrate the importance of choosing appropriate fault models. Such models are required in the testbed-driven performance and dependability evaluation of distributed systems. In such systems, common disturbances such as packet loss may have an adverse effect on both dependability and performance of the network, and thus on the system itself. As illustrated in e.g. [RvMW06b, RW08b], packet loss may have an impact on the dependability of higher networking and system layers even with the TCP protocol, which guarantees reliable data transmission. Consequently, methods for reproducing disturbances are required in order to study the dependability and performance of distributed systems in testbeds.

Injection of IP packet loss is an indispensible tool when evaluating fault-tolerant network protocols. However, fault injection at the IP level also provides a convenient way for assessing performance and reliability of distributed systems in which the network stack is considered just an off-the-shelf component. As the injected faults force the network stack to apply the same fault-handling procedures that are executed under real-world operating conditions (such as e.g. packet retransmissions in TCP), the same operational patterns of the network stack that are present in a real network can be expected to emerge in the testbed [RW08c].

As we have observed in Chapter 1, IP packet loss patterns have been the focus of intense study in the past decades. While in the simplest case packet loss may be described by a Bernoulli model, packet loss is often comprised of bursts of elevated loss probability, which can be modelled more closely with Gilbert-Elliot (GE) models [ZPS00, ZDPS01, HGHl08]. A Gilbert-Elliot model describes the loss process as a Markov Chain with different loss probabilities for each state.

Our focus in this chapter is on the application of fault-models in fault-injection driven performance and dependability studies of distributed systems. Considering three examples of typical scenarios that differ with respect to their communication patterns, we investigate the impact of the choice of fault-model on the results we obtain. The chapter is structured as follows: In Section 6.1 we describe common methods to measure and inject packet loss and introduce the three fault-injectors we compare. In Section 6.2 we evaluate the impact of faults injected with each of the three fault-injection methods on different types of arrival streams.

6.1 Measurement, Modelling, and Injection of IP Packet Loss

In order to inject realistic loss patterns, an understanding of loss patterns in real-world networks is required. Such an understanding can be gained by measurements on real networks. There are a number of methods for obtaining packet-loss traces: Measurements may be performed using monitoring approaches [CWA+05, HS08, HNA02], router measurements [BS03], and probes [ZPS00, ZDPS01]. Since monitoring and router measurements require a dedicated infrastructure, probes are often the preferred approach to gather large, representative data sets. Probe-based approaches typically consist of one host sending numbered probe messages to another host. The receiver either records their arrival or replies with an acknowledgment to the sender. Insight into the characteristics of the loss process is then obtained by an analysis of the recorded traces of sent and received packets. While implementation details may vary (e.g. one could use the Ping utility or the Zing utility [ZPS00] for different arrival processes), measurement studies using this methodology share the common property that the arrival process is independent of the loss process. That is, the frequency of probe packets is not influenced by packet loss.

6.1.1 Packet-Loss Models for Stochastic Fault-Injection

The result of a measurement study is typically a trace of the observed loss process. While such traces may be used for fault-injection by simply replaying them [KBF09], this method suffers from a number of shortcomings [RW08c]. In particular, sharing of traces is difficult due to their size and privacy issues, and traces can be neither generalised nor parameterised. Consequently, one typically derives models that describe important characteristics of the traces and uses these models in further evaluation steps. As we have seen in our survey in Chapter 1 (see Section 1.2.2.4 and Table 1.1), the literature provides two common models for the IP packet loss process, viz. Bernoulli loss models and Markovian loss models.

Bernoulli loss models describe the packet loss process as a sequence of Bernoulli trials with identical loss probability l. For each packet, a Bernoulli trial is performed,

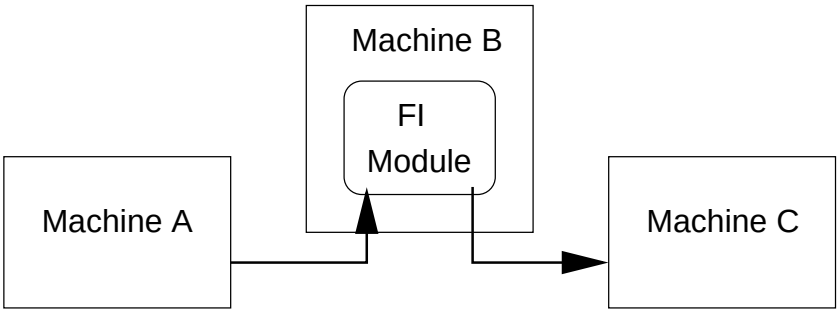


Figure 6.1: General approach to packet-level fault-injection.

and the packet is dropped according to the outcome of the trial. Since traces often show burstiness in the loss process, that is, interchanging periods of different loss rates [ZPS00, ZDPS01], *Markovian models* have been proposed as an alternative to the simple Bernoulli model. With Markovian models, the loss process is described by a Markov chain of length n whose states $s_1, \ldots, s_n$ correspond to Bernoulli trials with different loss probabilities $l_1, \ldots, l_n$. Packet traces are modelled by parameterising the model such that if reflects the distribution of the length of times for which loss levels have been stationary. In the simplest case, one can model lossy and loss-free periods [ZPS00] individually by a two-state Markov chain, i.e. assume loss probabilities $l_1 = 0$ and $l_2 = 1$ for the two states. However, larger models may capture the loss process better [ZDPS01]. By convention, Markovian loss models are often called (extended) Gilbert or Gilbert-Elliot models. A good overview of such models is given in [HGHl08].

The most straightforward way of injecting packet loss according to a loss process described by a loss model is depicted in Figure 6.1: One computer is set up to act as a router between two or more hosts or networks. On this machine, the TCP/IP stack is augmented by a fault-injection (FI) module. For each arriving outside packet the FI module determines whether to forward the packet to its destination or to discard it. With a Bernoulli model, for each packet one trial is performed and the packet is dropped according to the outcome of this trial. With a Markovian model, the dropping probability for the Bernoulli trial is selected according to the current state of the model when the packet arrives.

6.1.2 The Problem with Markovian Models in Fault-Injection

Note that in the above we have omitted describing when state-changes occur in the fault-injector's Gilbert-Elliot model. In fact, so far we have intentionally refrained from mentioning the time domain for Markovian models at all.

In the literature (e.g. [HGHl08]), Gilbert models are defined as embedded Discrete-Time Markov Chains (DTMCs). The left part of Figure 6.2 shows an example of a

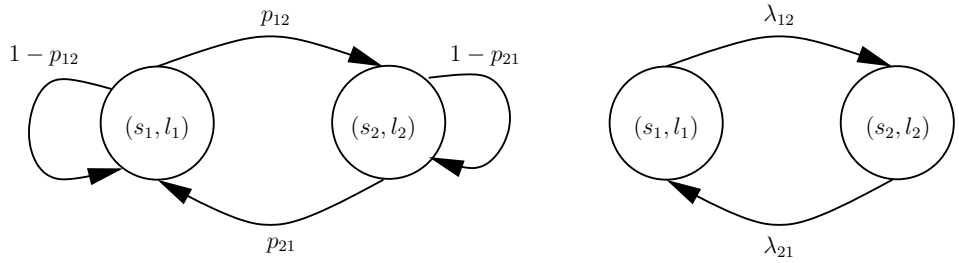


Figure 6.2: Gilbert model as a DTMC (left) and CTMC (right).

two-state discrete-time Markovian loss model with loss probabilities l_1 and l_2 and state transition probabilities p_{12} and p_{21}. In a DTMC loss model, state changes occur at discrete time intervals. Certainly, this view is appropriate for modelling the loss process as described by a packet trace: Each packet constitutes a time instant at which a state-change may occur. Implementation of fault-injection using a discrete-time model is straightforward as well: With each arriving packet the next state is selected randomly from the possible successor states (including the same state) of the current state.

An alternative definition of the Gilbert-Elliot model views the model as a Continuous-Time Markov Chain (CTMC). In this view, state sojourn times are described by exponentially distributed random variables. The right-hand side of Figure 6.2 illustrates a two-state continuous-time Markovian loss model, again with loss rates l_1, l_2. The transition rates in the CTMC model are given by λ_{12} and λ_{21}. Packet-loss injection according to a continuous-time Gilbert model is implemented by playing the CTMC, as follows: Upon entering a state, the state sojourn time is drawn from an exponential distribution with rate equal to the sum of outgoing rates. When this time has passed, the next state is chosen based on the embedded Markov chain.

Let us now consider the relation of these different formulations of the loss model. With respect to *describing* packet-loss traces, the CTMC model is obviously equivalent to the embedded DTMC model, since both can capture interchanging periods of different loss levels. With respect to *generating* packet loss, however, we note a difference: In the embedded DTMC model, state-changes in the model can only occur upon arrival of packets, i.e., the packet-loss process is dependent on the arrival process. This is in contrast to the CTMC model, where state-changes occur independently, and requires a closer look at the arrival process. As described above, packet-loss measurements are typically obtained using an arrival process whose packet interarrival times are independent of packet transmissions. While there exist network protocols where this is the case, most protocols adapt their sending rate based on the success of previous packet transmissions. In particular, the Transmission Control Protocol (TCP) employs various mechanisms to reduce the sending rate if packet

loss occurs. These reductions can be quite drastic, especially during the connection-setup phase, where the retransmission interval, starting at 3 s, is doubled for each timeout (i.e. packet loss) [KR01]. If such a reactive protocol is studied using fault-injection with a DTMC Gilbert model, the protocol's sending rate is affected by the loss process, while at the same time the loss process depends on the sending rate. This observation raises the question whether the obtained results are sound.

6.2 Experiments with IP-Packet-Loss Models

We now investigate the question in an experimental study of different fault models in three example scenarios. All scenarios use fault-injection to assess the performance of a distributed system with two hosts communicating over a lossy link. The scenarios differ in the characteristics of the arrival process of the network traffic. We consider three arrival processes:

Independent Arrival Stream First, we consider a scenario where one host sends messages at a constant rate, thereby generating an arrival stream that is independent of the loss process. Examples for this type of traffic pattern may be heartbeats, periodic log messages, or clock frequency synchronisation as used in PTP [IEE08a]. Note that, while such a protocol might employ acknowledgments and retransmissions of lost packets to ensure data transmission, retransmissions are assumed to take place at the time a normal packet transmission would have happened. That is, in this scenario retransmissions due to packet loss do not affect the arrival process observed by the network.

Long TCP Arrival Stream Second, we study the performance of a large data upload over a single long-lived TCP connection. Since TCP adapts its sending rate dynamically based on the available bandwidth, using, among others, packet loss events as indicators of network overload [KR01], in this scenario the arrival process is not independent of the loss process.

Short TCP Arrival Stream In our third scenario we consider the performance of HTTP when using TCP connections over a lossy link. This scenario differs from the previous one with respect to the length of the connections and the amount of data to be transmitted. Here, we assume short connections, such as may be observed when downloading small web pages or in client-server communication in Web Services scenarios.

Packet loss was injected according to the Bernoulli, discrete-time Markovian and continuous-time Markovian models. We employ the following fault-injector implementations:

Netem (Bernoulli Packet Loss) The Linux operating system supplies the kernel module Netem for injecting various disturbances at the IP level. In particular, this module supports the Bernoulli loss model, which can be parameterised by providing the desired loss rate.

Netem CLG (Discrete-Time Markovian) The NetEm-CLG (Correlated Loss Generator) module [SLO10] extends the default NetEm implementation by a discrete-time extended Gilbert-Elliot model. The module supports up to four states with fixed transition structure. In the following we use it to implement a two-state Gilbert model.

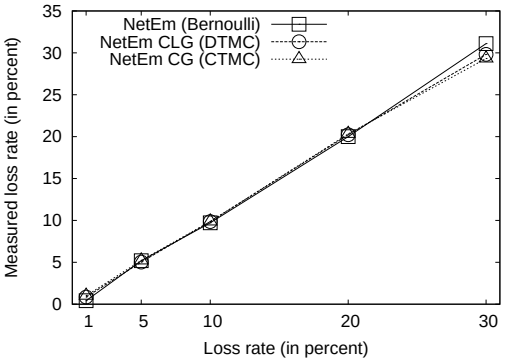
Netem CG (Continuous-Time Markovian) We developed the Netem-Continuous-Gilbert (NetemCG) module that implements continuous-time Markovian loss models as an extension to the default Netem module of the Linux kernel. The module support continuous-time Gilbert models with an arbitrary number of states and an arbitrary structure and is parameterised by a text file. Furthere Details on the implementation of the module are available in [Drä11, RDW11]. As with the NetEm-CLG module, we configure NetEm-CG such that it provides a two-state Gilbert model.

6.2.1 Experiment Setup

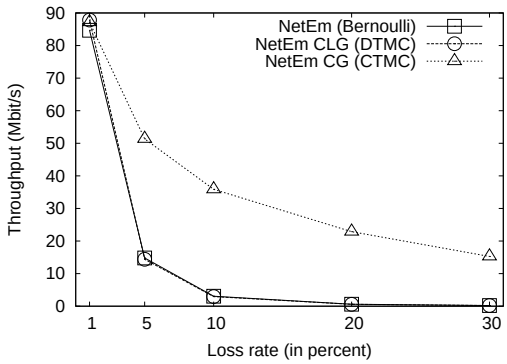
We use three Linux machines to set up the experiment environment shown in Figure 6.1. Machines A and C run Linux kernel 2.6.26, while machine B runs 2.6.37. We routed packets from A to C over B, while packets from C to A were transmitted directly, in order to ensure that the chosen packet-loss rate is achieved.

The arrival streams required by our scenarios are generated as follows: For the independent arrival stream, we use the Ping utility to send packets with a constant interarrival time of 200 ms. Since Ping does not retransmit lost packets, this reflects the scenario where the packet arrival process is independent of the loss process. Ping sends a stream of ICMP packets. For each packet, the recipient replies with another ICMP packet. Upon receipt, the original sender prints out the sequence number of each received packet. We run the Ping test for 30 min and record the sequence numbers.

We emulate a long TCP connection using the TCP_STREAM test of Netperf 2.4.5 [Jon09]. In our experiments, Netperf generates a unidirectional TCP stream from machine A to machine C and measures overall throughput as well as averages taken over time-windows of approximatel 10 s length. The test duration was originally set to 5 min. Due to our observation that for higher loss rates the test failed to transmit a sufficient amount of data within 5 min to get a measurement of the



(a) Packet loss rates in the Ping test (independent arrival stream).



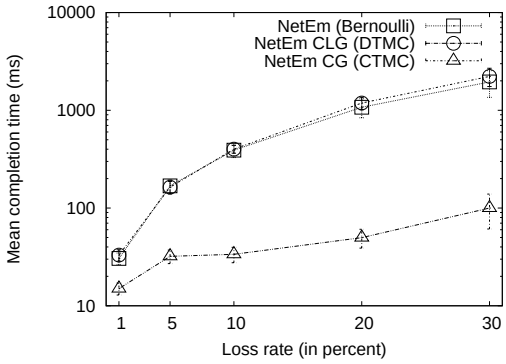
(b) Average throughput in a long-lived TCP connection.

Figure 6.3: Measurement results for the independent Ping arrival stream and the TCP arrival stream.

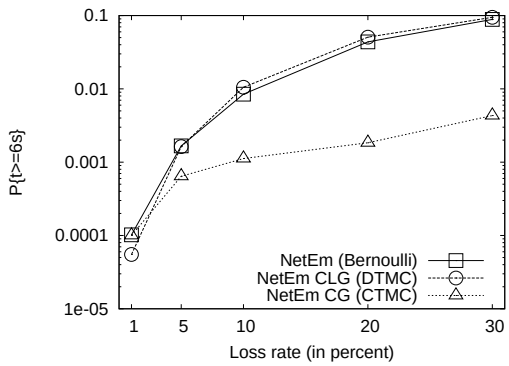
throughput, we also ran experiments where we set the test duration by specifying a limit on the amount of data to be transmitted instead (100 MB).

Performance of short HTTP connections was studied using the Apache JMeter tool [The08] and the DHTTPD/1.02a web server [Klo11]. We set up the web server on machine C and configured JMeter to repeatedly download the server's default homepage without any embedded content. In our setup, the default homepage is a single static HTML file of 5258 bytes length; consequently, we expect the processing overhead of the web server to be negligible. We configured JMeter such that the KeepAlive feature of HTTP was not used. Combined with the small file size this results in very short TCP connections, consisting almost entirely of connection setup and connection teardown. We originally ran the test for 10 min, but increased the time to 30 min for higher loss rates with the discrete-time loss model, in order to obtain at least 500 samples for each loss rate.

The loss-injector modules are loaded in machine B. We parameterise the models such that they produce packet-loss rates of $l = 0.01, 0.05, 0.1, 0.2,$ and 0.3. For the Bernoulli model, parameterisation consists in setting the loss rate to the desired value. For the NetEm-CLG module we specify the desired loss rate l of the model. The model is then parameterised such that $p_{12} = l$, $p_{21} = 1 - p_{12}$, $l_1 = 0$ and $l_2 = 1$. With the NetEm-CG module we use the same loss rates l_1, l_2, set $\lambda_{12} = 1$, and choose λ_{21} such that the loss rate l is achieved. Sojourn times were scaled to give a mean sojourn time for the loss-free state of 100 ms.



(a) Mean HTTP download times (log scale, with 95% confidence intervals).



(b) Probability of HTTP connection times larger than 6 s.

Figure 6.4: Measurement results for short TCP connections.

6.2.2 Results

Figures 6.3(a) through 6.4(a) show the average loss rate, average throughput, and mean HTTP download times obtained using Ping, Netperf and JMeter, respectively. Considering first the loss rate (Figure 6.3(a)), we observe that all fault injectors generated the desired loss rates. This demonstrates that, with respect to the loss rate, the fault model implementations are equivalent when an independent arrival process is used. Turning towards throughput measurements (Figure 6.3(b)), however, we note that with the Bernoulli model and with the DTMC model performance drops much faster than with the CTMC model. Finally, consider the mean HTTP download times shown in Figure 6.4(a). Here, a similar tendency can be observed: Packet loss generated using the CTMC model has a much lower impact on download times than Bernoulli and DTMC Gilbert-model loss.

The last scenario is especially useful for explaining the difference in the results, as the short duration of individual HTTP transmissions implies that each connection consists almost entirely of the TCP three-way handshake in the connection setup phase. In this phase, TCP detects packet loss by the RTO timeout, which starts at 3 s. That is, if the initial SYN packet is lost, the first retransmission happens after 3 s. For each timeout event, the RTO is doubled, i.e. the next retransmission occurs at 6 s, and so on [KR01]. Since the delay in our network is negligible, it is reasonable to assume that samples larger than or equal to 6 s have been affected by at least two packet losses in the connection-setup phase. From Figure 6.4(b) we see that the probability of such samples grows with the loss rate for all models. More important, however, is that the discrete-time Gilbert model results in a much higher probability of two-packet losses during TCP connection setup than the continuous-time model. This can be attributed directly to the discrete-time loss process: The model can

only leave the loss state upon packet arrivals. That is, if the model is in the loss state after dropping one packet, it will still be in the loss state when the next packet arrives, irrespective of the interarrival time between both packets. In contrast, the continuous-time model changes its state independently of packet arrivals and may thus be in another state when the next packet (or retransmission) arrives.

6.3 Concluding Discussion

In this chapter we have investigated the impact of the interaction between the arrival stream and the fault model used in fault-injection studies. We observed that with dependent arrival streams as generated by TCP the results obtained using discrete-time and continuous-time Markovian loss models differ significantly, with discrete-time models resulting in much lower performance. When used in a performance study, the two models may lead to different conclusions. With a discrete-time model, where changes occur only at packet arrivals, very short retransmission timeouts may improve performance significantly, as they increase the rate of state transitions, and thus allow the model to reach the loss-free state earlier. With a continuous-time model, shorter timeouts might still improve performance marginally (due to the fact that the loss-free period is detected earlier), but will not have as big an impact, since the loss model changes state independently of packet arrivals. On the other hand, the increase in packet transmissions will increase the load and the effective packet loss rate.

The goal of this chapter was to highlight the importance of choosing the right fault model, and thus we cannot conclude the chapter with a clear recommendation which model one should use to reflect reality in a fault-injection experiment. Such a recommendation needs to be based on knowledge of whether the packet-loss process is independent of the arrival process. Currently available measurement studies of packet loss only provide traces which can be described equally well by both models, and give no indication with respect to dependence or independence of the loss process. Investigating this question is an important open question. However, except for special situations (e.g. where the arrival process saturates the link entirely) it appears unlikely that the loss process should be governed by the arrival process alone, so we do provide a tentative recommendation to use CTMC models.

Chapter Seven

Phase-type Distributions in System Evaluation

In Part II we focussed on phase-type distributions as a valuable tool for describing real-world phenomena for system evaluation using test-beds, simulation, and analysis. We gave algorithms for efficient random-variate generation, developed optimisation methods that reduce the cost of random-variate generation, and described approaches for fitting phase-type distributions to empirical data.

In this chapter we demonstrate the application of phase-type distributions in accurate and efficient system evaluation. Starting with the first publications on the topic, PH distributions have been well-established in analytical approaches (e.g. [Neu81]) to system evaluation, while simulation-based approaches are relatively un-explored to this day. Our main focus in this chapter is therefore on simulation-based applications. We first introduce the `libphprng` library for efficient random-variate generation from PH distributions. We then present two case-studies where the library is used. The case-studies are selected to illustrate the application of phase-type models and the improvements that can be obtained in terms of accuracy of the results and efficiency of the evaluation.

Our case studies cover quite different fields. The first case study illustrates the use of PH distributions in the simulation-based evaluation of restart algorithms. Here, response-times are considered i.i.d. random variables for which a distribution is given. The second case-study illustrates how PH distributions may help to improve model scalability when we study frequency synchronisation in mobile-backhaul-networks. We show that by applying phase-type distributions in component-wise abstraction we can substantially reduce simulation times for large models.

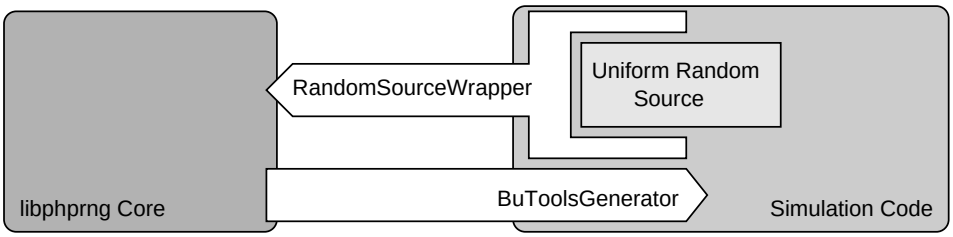


Figure 7.1: Architecture of the libphprng library

7.1 The libphprng Library

Chapter 3 introduced several algorithms for efficient random-variate generation from phase-type distributions. In order to enable the application of these algorithms by non-expert users, we developed the libphprng library. Design goals for the library were ease of use and good integration with a variety of discrete-event simulators. Libphprng is part of the Butools package [BBH+11], which provides easy-to-use implementations of various methods for Markovian and non-Markovian distributions and processes.

Libphprng is a shared library that implements a C++ class for random-variate generation from phase-type distributions and provides a generic interface for connecting the library to arbitrary simulation tools. Figure 7.1 shows the basic architecture of libphprng: The library core provides the functionality for generating PH-distributed random variates. This code is accessible through the `BuToolsGenerator` class. The user creates an object of this class for each required PH-distributed random-variate source, specifying a file that contains the parameters of the distribution. Upon creation, a `BuToolsGenerator` object reads the PH distribution from the file and parameterises itself such that it generates random variates from this distribution. Then, each call to the method `BuToolsGenerator::getVariate()` yields one random variate.

The code for generating random variates requires uniform random numbers. OMNeT++ and other discrete-event simulators provide elaborate uniform random number generators. In particular, OMNeT++ offers several user-configurable independent random-number streams. These facilities are supported by libphprng through the use of a thin, simulator-specific wrapper module for the simulator's native random number generator. The user needs to instantiate an object of a suitable `UniformRandomWrapper` class and register this object with the `BuToolsGenerator` object. The `BuToolsGenerator` object will then use the specified uniform random number generator, supporting simulator-specific features such as independent streams. We provide wrapper code for both OMNeT++ and ns-2; other wrappers can easily be built following these examples.

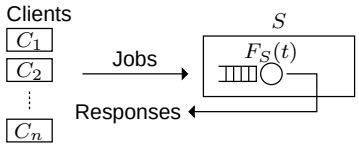


Figure 7.2: System Model for Restart

7.1.1 Related Work

The only other tool that that supports phase-type distributions in discrete-event simulators is the `ArrivalProcess` module by Kriege et al. [KB11]. The module provides stochastic processes for OMNeT++. In particular, `ArrivalProcess` supports Markovian Arrival Processes (MAPs), of which phase-type distributions are a sub-class. With proper input, `ArrivalProcess` can be used to generate phase-type distributed random variates. `ArrivalProcess` generates random variates from Markovian arrival processes and PH distributions by 'playing' the Markov chain.

`ArrivalProcess` is capable of generating random variates from more elaborate stochastic processes than libphprng. For instance, `ArrivalProcess` can be used to generate correlated random variates, which is not possible with libphprng. On the other hand, the limitation of libphprng to phase-type distributions enables much more efficient algorithms, as structural properties, sub-classes, and optimisation can be used. Such methods are not known for more general stochastic processes. We will illustrate the performance advantage of libphprng in Section 7.2.4. Furthermore, from an application point of view, libphprng is more general, in that it is completely independent of the simulator. Since it is implemented as a library, libphprng can be linked dynamically and seamlessly to any simulator (or other application requiring PH random variates), thus enabling efficient random-variate generation without changes to the library and with minimal changes to simulation code.

7.2 Case-Study I: Accurate Evaluation of Restart Strategies

The first case-study applies the libphprng library in an evaluation of restart using the discrete-event simulator OMNeT++ [Var01]. This case-study was selected to illustrate the application of the library in a simulation and the increase in accuracy of the simulation results that can be obtained by using a phase-type model. This case-study is similar to that in [RW10], however, since the goal here is to illustrate the use of libphprng, we simplify the study considerably and focus on methodological and implementation details.

7.2.1 Background

Consider the problem of developing an algorithm for computing the request timeout in a service-oriented system. When this timeout elapses, the client assumes that the request failed or will take very long to complete. In this case, the client aborts and restarts its request. If failures and long completion-times are caused by transient faults, the restarted request may yield a response that arrives sooner than if the client had waited for the response to the original request. In service-oriented systems, such transient faults can be due to common influences such as IP packet loss and temporary server overload [RvMW04, RWW09]. In these situations, response-times may often be reduced by restart. Restarting too soon, or too often, however, increases load and may thus exacerbate the problem, increasing response-times even further. Therefore, the optimal restart interval must be computed based on operating conditions.

While there exist analytical approaches for computing the optimal timeout in simple scenarios [vMW06], more complex service strategies or job semantics as well as scenarios with multiple clients competing for the same resources require simulation.

In our example we consider the following scenario (Figure 7.2): Clients generate jobs according to a Poisson process. Each client has an internal FIFO queue from which it removes jobs one-by-one and submits them to the server for processing. If the job is not finished before the restart timeout, the client aborts the job and immediately re-submits it to the server. The next job is taken from the queue only after the current job has been completed. In the server, incoming jobs are stored in one single FIFO queue and processed on a first-come, first-served basis. After the server completes a job, it sends back a response to the client. The time for job processing is governed by the service-time distribution F_S. We are interested in whether restart may help in this scenario as well as in the optimal restart timeout. For simplicity, in the following we assume that there is only a single client. In this case, no queueing occurs in the server.

7.2.2 Application of libphprng

We implemented this scenario as a simple simulation in OMNeT++. In order to use the libphprng library within a simulation, two steps are required:

1. The simulation source code needs to be modified such that the `BuTools Generator` object from the libphprng library is used.

2. The simulation must be linked with the `libphprng.so.1` and `libOMNet RandomSourceWrapper.so.1` libraries.

```
class WorkServer: public cSimpleModule {
        /* ... */
        OMNetRandomSourceWrapper * orsw;
        PhGen * phgen;

        private void initServiceTime
                        (char* filename);
        private double getServiceTime();
}

void WorkServer::initServiceTime
                (char* filename) {
        phgen = new BuToolsGenerator(filename)
        orsw = new
                OMNetRandomSourceWrapper(2);
        phgen->setUniformRandomSource(orsw);
}

double WorkServer::getServiceTime() {
        return phgen->getVariate();
}

/* ... */
```

Figure 7.3: Source code for the inclusion of libphprng in a simple OMNeT++ server class.

Figure 7.3 illustrates the modifications required for using the libphprng library in our example. The figure shows the relevant parts of the `WorkServer` class. In our simulation, the `WorkServer` class implements the simple server. The method `initServiceTime()` is called during initialisation of a `WorkServer` instance to initialise the service time distribution. During operation, every time a job from the queue enters service or is restarted, a service-time sample is required to parameterise the delay. For each service-time sample, `WorkServer` calls its `getServiceTime()` method.

The first modification affects `initServiceTime()`: The server class uses a single `BuToolsGenerator` object to store the service-time distribution. This object is initialised in the method `initServiceTime()`, as follows: First, the new object is instantiated. Upon instantiation, it reads the distribution from the given filen. Second, a wrapper object for the uniform random number stream with index 2 is

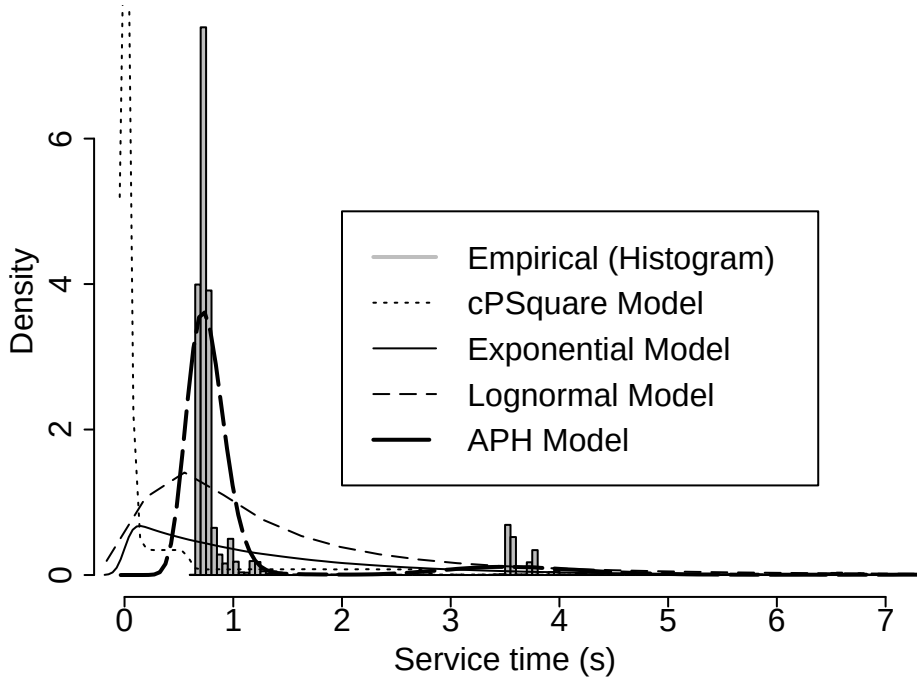


Figure 7.4: Histogram of empirical service-times and approximated service-time densities (truncated at 7 s).

created, and, third, the wrapper object is registered with the `BuToolsGenerator` object.

During operation, calls to `getServiceTime()` must return service times from the service-time distribution. With each call, `getServiceTime()` draws a PH-distributed random variate from the PH distribution stored in `phgen`, by calling the `getVariate()` method.

7.2.3 Evaluation

We will now illustrate the advantage of using phase-type distributions in a simulation-based evaluation of the impact of the restart interval on the response-time. We consider one client and vary the restart interval from 0.025 s to 60 s. For each restart interval, we observe response-times and compute the mean. We are interested in whether there is an optimal restart timeout, and in its value.

We use a job interarrival time of 10 s and base the service-time distribution F_S on response-time measurements that were obtained in a test-bed with injected packet

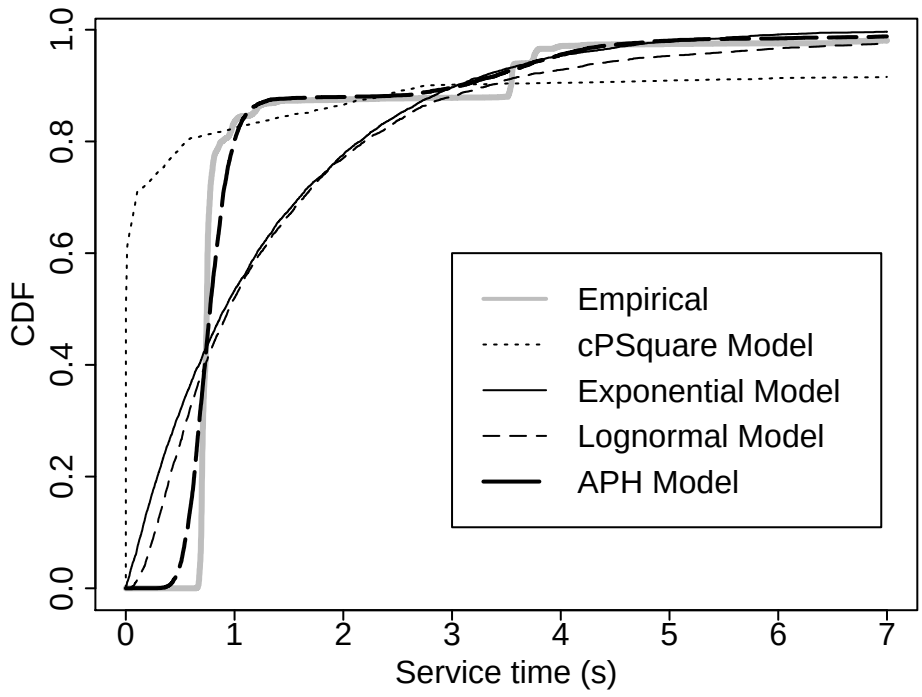


Figure 7.5: Empirical distribution functions of the empirical and approximated service-time distributions (truncated at 7 s).

loss. We compare the results obtained using four different models for the service-times: First, we use the empirical distribution of the samples to generate random variates. In the following, this model is called *cPSquare Model*. Second, for the *Exponential Model* we use an exponential distribution, parameterised such that the mean is equal to the mean of the data set. Third, we use a *Lognormal* distribution that we fitted to the data set using the R statistical tool [R D06] such that it reflects the mean and the variance well. Finally, the *APH Model* is an acyclic phase-type distribution of size 50 fitted using the PhFit tool [HT02].

The cPSquare model is implemented using the `random()` method of OMNeT++'s `cPSquare` class. In the initialisation routine of the server module we create an object of this class and import our data set into the object. The Exponential and Lognormal models use the respective functions of OMNeT++'s library of continuous distribution. The APH model employs the libphprng library, as described above.

First, we consider how well the models represent the measurement data. Using our simulation, we computed 10^6 response-time samples from each model. Figure 7.4

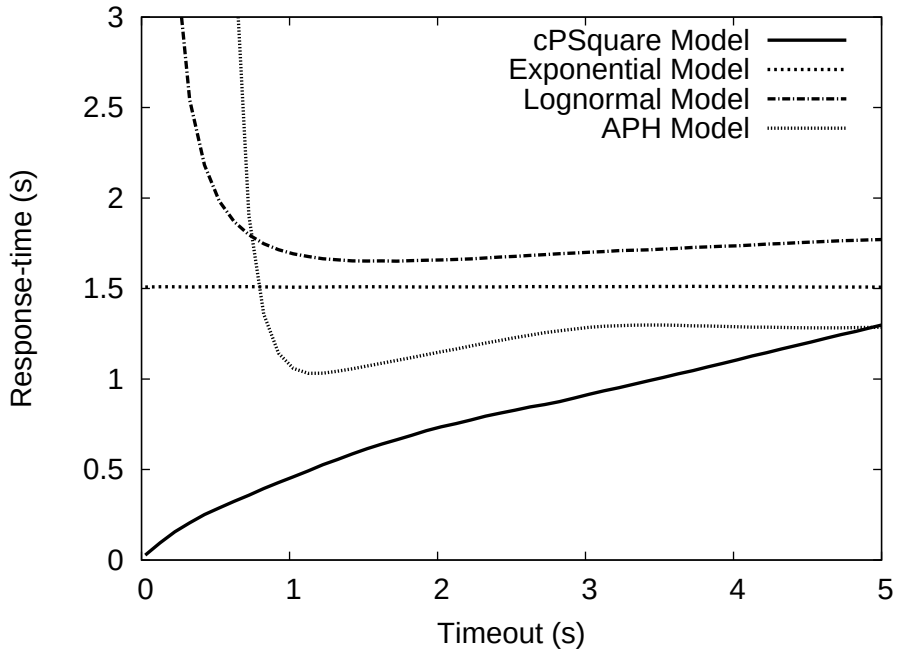


Figure 7.6: Mean response-times for different timeouts, using different service-time distributions.

shows the densities of the empirical data and of the models. Note that the empirical density has two clusters of data, one close to 1 s, and the other close to 4 s. From our experiment setup we know that the first cluster corresponds to requests where no IP packet loss occurred, while requests in the second cluster have been delayed by 3 s by the TCP RTO (cf. [KR01, RvMW04]). Observe that only the APH model shows both of these spikes, while the other models do not fit the empirical density well. In particular, the cPSquare model underestimates service-times, while the Exponential and Lognormal models do not represent the clusters at all. However, the Lognormal model captures the variance of the distribution. The differences become more evident in Figure 7.5, where we show the cumulative distribution functions (CDFs) of the data sets: The only model that reflects the steps in the empirical distribution is the acyclic phase-type distribution.

Figure 7.6 shows mean response-times observed for different timeout intervals using these service-time models. Note that there are large differences between the response-time curves: For the cPSquare model, mean response-times grow monotonously with increasing timeout. This implies that the optimal timeout is at zero. Immediate restart, however, is certainly not feasible, and thus results with the cPSquare model imply that restart should not be applied. A similar conclusion can

be drawn from the Exponential model, where restart does not change the mean response-time at all. With the Lognormal model, we observe that very low restart intervals increase the mean response time drastically, and that the mean response time first drops and then grows slowly as the timeout value increases. While results with this model do not clearly indicate the existence and location of an optimum, they do show that the timeout should not be below 1 s. Finally, with the APH model we clearly observe that there is an optimum around 1.25 s. This observation is corroborated by the result of computing the optimal timeout using the algorithm from [vMW06].

Our observations highlight the need for accurate models. In our case study we wanted to obtain insights into the applicability of restart in a scenario where service-times are distributed as in our measurements. The cPSquare model implies that restart should not be applied. Taking into account the bad fit between the empirical distribution and the cPSquare model, we should not trust this result. The Exponential model implies that restart neither helps nor hurts. However, this result has no bearing on the scenario at hand at all: As shown in [Wol10], the memoryless property of the exponential distribution means that restart has no effect on an exponential distribution. Consequently, irrespective of the parameterisation, any exponential distribution would have yielded the same result. In contrast, the simulations with the Lognormal model point towards the existence of an optimal timeout. Since the applicability of restart strongly depends on the variance of the distribution [Wol10], and the Lognormal model captures the variance well, we may trust this result more than those from the first two models. Finally, the experiments with the APH model, which captures the shapes of the density and distribution well, show both the existence of an optimal timeout and its location.

7.2.4 Performance of the libphprng Library

In this section we consider the computational demands of generating random variates. We demonstrate that the generation of random variates can take a significant portion of the total simulation time. Since drawing PH distributed random variates requires more effort than drawing basic random variates (like exponential, normal, etc.), it is necessary to put emphasis on the efficiency of these methods to reduce the simulation time. As mentioned earlier, libphprng is able to exploit the special structural properties of the input in order to achieve better performance, but we also show that our implementation is efficient with arbitrary input (exhibiting no supported structure) as well.

We compare our results to the `ArrivalProcess` object proposed by Kriege and Buchholz in [KB11]. The `ArrivalProcess` object can generate random variates from Markovian Arrival Processes (MAPs), which are a superclass of phase-type distributions, and thus our comparison is not entirely fair, since `ArrivalProcess`

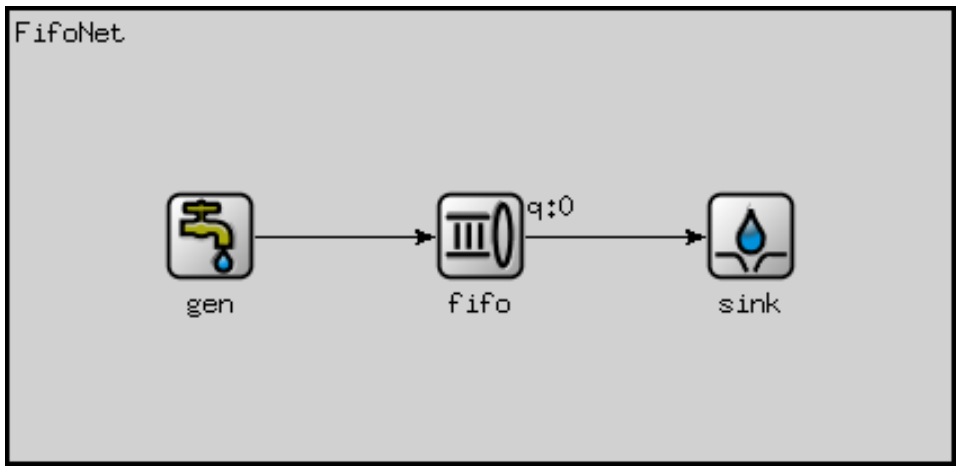


Figure 7.7: Queueing-Model used in the Performance Evaluation

supports a much larger set of stochastic processes than libphprng. It will be obvious, however, that in situations where PH distributions are sufficient to capture the properties of the phenomenon, a much more efficient implementation can be produced.

In our performance evaluation we use the simple model depicted in Figure 7.7. It consists of a source that generates messages with a deterministic interarrival time. The messages enter an infinite queue that serves them according to a FIFO discipline. The service times are PH-distributed random variables. After service, messages are transferred to the sink, where they are destroyed. To make the queue busy the deterministic interarrival time of the queue has been set to obtain a utilisation of 96%.

In the first experiment the input PH distribution has no supported special structure and a dense sub-generator matrix $\mathbf{Q}$. The distribution in this experiment consists of 20 states. We picked the entries of both the generator matrix and the initial vector randomly, ensuring that $(\boldsymbol{\alpha}, \mathbf{Q})$ is a phase-type distribution. The simulation ends after 30 sec of CPU time. We consider both the simulation speed, as reported by OMNeT++, and the portion of CPU time that was spent in the random-variate generator, according to `callgrind` from the Valgrind package [NS07].

We summarise our results in Table 7.1. Note that both simulations spend a significant amount of time in the random-variate generation routines. This result makes it obvious that it is definitely worth to develop and use efficient random-variate generation methods, since these methods can dominate the execution time when there are lots of random events in the model. Second, Table 7.1 shows that our library is almost 50% faster than the general `ArrivalProcess` module.

Method	Simulation speed	Portion of sim. time
libphprng	359,364 events/sec	93%
`ArrivalProcess` [KB11]	247,159 events/sec	95%

Table 7.1: Simulation performance with random PH-distribution.

Method	Simulation speed	Portion of sim. time
Exponential	5,473,758 events/sec	7.5%
Lognormal	4,907,375 events/sec	12.5%
libphprng	1,598,788 events/sec	70%
`ArrivalProcess` [KB11]	397,440 events/sec	93%

Table 7.2: Simulation performance for models from Section 7.2.3.

In our second experiment we compare the speed of random-variate generation from the different models considered in Section 7.2.3. We employ the Exponential, Lognormal, and APH models. The APH model is simulated using both the `ArrivalProcess` module and libphprng. In contrast to `ArrivalProcess`, libphprng can make use of the fact that the model is an APH distribution in bi-diagonal form and thus has a special structure that allows both optimisation and efficient random-variate generation using the algorithms described in Chapters 3 and 4.

The results of the second experiment are shown in Table 7.2. First, observe that the two phase-type generators are much slower than random-variate generation from an exponential or lognormal distribution. Thus, better representation of the phenomenon under study (cf. Section 7.2.3) is bought at the cost of increased simulation time. This highlights the importance of efficient methods for PH-random-variate generation. Second, note that in this experiment our libphprng library is roughly four times as fast as `ArrivalProcess`. This result highlights the improvement in efficiency that is made possible by the fact that libphprng can make use of the special structures available for APH distributions.

7.3 Case-Study II: Evaluation of PTP Performance in Tree-Structured Networks

The second case-study shows the use of the libphprng library in hybrid models for scalable discrete event simulation, as proposed in [BCH+12]. Hybrid models aim to improve the efficiency and scalability of discrete-event simulation through replacement of detailed, inefficient components of the simulation model by abstract

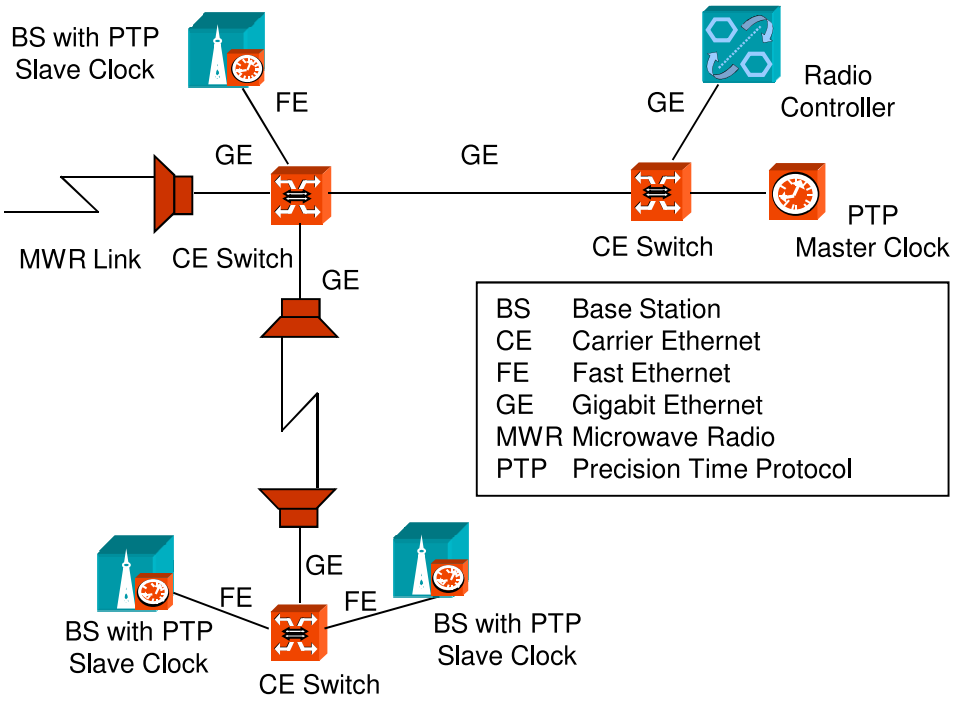


Figure 7.8: Network using synchronisation of base station with ToP using PTP. The master clock (in the upper right corner) sends PTP Sync messages to the slaves. Note that in this application domain, network traffic is usually shown as flowing from right to left.

stochastic models of the components. The case-study we describe here uses phase-type approximations for detailed model components. Phase-type random variates are generated using the libphprng library.

We start with an overview of the technical background and discuss the importance of highly-detailed models for reproducing the phenomenon under study. We then show how the scalability issues of highly-detailed models can be overcome using PH distributions.

7.3.1 Technical Background

Backhaul networks for GSM, WCDMA and LTE have very strict requirements for the frequency on the air interface of the base stations. If these requirements are violated, seamless handover between neighbouring base stations cannot be guaranteed anymore. As local oscillators cannot guarantee the required accuracy, frequency synchronisation (also referred to as syntonisation) of base-station clocks (called

slave clocks) to a high-quality primary reference clock (master clock) is mandatory.[1] Whereas current synchronous TDM-based networks can convey the frequency of the master clock to the slaves via the bit clock of the network itself, this becomes impossible with the migration of backhaul networks to asynchronous packet-switching networks (e.g. Carrier Ethernet, CE). In such networks, timing-over-packet (ToP) using the precision time protocol (PTP) according to IEEE standard 1588-2008 [IEE08a] is the preferred syntonisation method.

The accuracy attainable by PTP is limited by variation in packet transmission delays, which in turn depends on a variety of factors, including the networking components, network topology and background traffic. Whether a network design will be able to sustain syntonisation using PTP is an important question in engineering backhaul networks. Black-box measurements on real hardware (e.g. [Cos08, Cos09, HSXX, AWGG04, ZHB10, Job09], see also [ITU08]) provide some insight into the behaviour of specific set-ups, but are expensive and cannot be generalised to large-scale systems.

The operation of PTP can be illustrated using the scenario shown in Figure 7.8: A PTP master clock syntonises PTP slave clocks in a single-ended way. The network has one PTP master clock on the right-hand side and several PTP slave clocks. At constant time intervals, the PTP master clock sends Sync messages to the slaves. The slaves use the interarrival time of consecutive Sync messages to synchronise their local clock frequency to that of the master clock. Network connections in this scenario can be either optical fiber Gigabit Ethernet (GE) links connecting carrier-ethernet (CE) switches, or microwave radio (MWR) links with radio antennas on either end. The last link to the mobile base station (BS) is usually a fast ethernet (FE) connection. Some PTP slaves are only a few links away from the master clock, but others can be at a distance of up to 20 links.

The Packet Transfer Delay (PTD) of the ith Sync message is defined as the difference between the time the message was received and the time the message was sent:

$$PTD_i = t_i^{received} - t_i^{sent}.$$

In the following, let PTD denote the packet transfer delay distribution, defined on $\mathbb{R}^+ \cup \{0\}$. Ignoring packet loss, the PTD of a network is bounded to an interval $[PTD_{min}, PTD_{max}]$ of the best and worst case transmission delay. In practice, PTD_{min} is the time required by a Sync message when there is no background traffic. PTD_{min} thus depends solely on the networking hardware, and $PTD_{min} > 0$. If there is background traffic, Sync messages may be delayed by processing of background packets. Then, $PTD_{max} > PTD_{min}$.

Syntonisation accuracy depends only on the variation in PTD samples, but not on the constant offset PTD_{min}. The variation is described by the Packet Delay Vari-

[1] In contrast, time and phase synchronisation are less important.

ation (PDV), defined as the shifted PTD distribution (section 6.2.4.1 in [ITU02])[2]:

$$PDV := PTD - PTD_{min}.$$

In the following we consider two ways of characterising PDV. A simple measure is given by the peak-to-peak PDV (p2pPDV),

$$p2pPDV := PTD_{max} - PTD_{min} = PDV_{max},$$

which gives the maximum PDV value. In practice, slave clocks typically filter out samples from slow PTP packets, as slow packets are likely to increase PDV. Slow packets are easily identified by computing the difference between timestamps set by the sender and receiver. The slave clock then uses only the fastest packets, with typical thresholds set at the 1% quantile of the PDV distribution or below. The quantiles of the PDV distribution thus provide a practical measure for PDV. Note that this measure is equal to the p2pPDV of the PDV distribution truncated at the respective quantile.

The upper PDV bound tolerable by a PTP slave clock depends on the PTP implementation and the accuracy of the local oscillators. These properties vary between vendors and are often considered proprietary information. However, a 1% quantile PDV between master and slave of 216 μs can be considered a reasonable target. Although we cannot embark on a detailed discussion of the implementation of a slave clock here, we note that slave clocks average PTD values over a certain time period and that the upper bound of 216 μs corresponds to a maximum deviation of 15 ppb with integration window size 4 hours.

7.3.2 PDV Characteristics

The characteristics of the packet delay variation encountered in a network depend on the networking hardware, the path length, and the background traffic load. As evidenced by measurement studies [Job09, Cos08, Cos09, HSXX, DHK+01, AWGG04, ZHB10, BGNV09], these parameters may result in a wide variety of phenomena. In order to be useful, our simulation models must be able to reproduce these effects. In the following we focus on the PDV of a single switch, as this will give us the basic building block of models for networks with longer paths (i.e. multiple switches) between master and slave clock. We assume that the switch is configured such that PTP packets have highest priority, because this configuration is common practice when engineering backhaul networks for syntonisation using PTP.

With these prerequisites we find two patterns in PDV measurements. Abstracting from device-specific constant offsets or scaling factors, these patterns can be

[2]Note that there exist different terminologies and definitions for the variation in transmission delays (sometimes also referred to as jitter), e.g. the instantaneous PDV, as defined by [DC02].

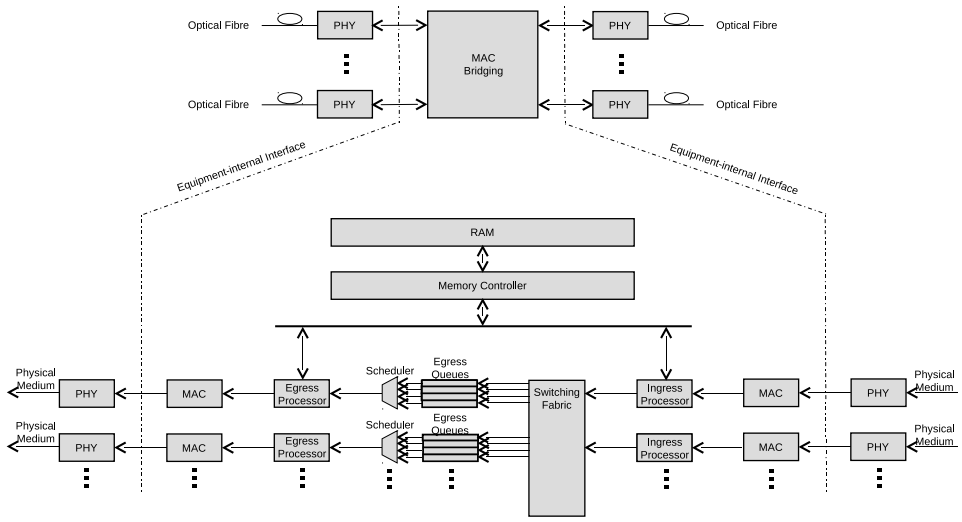


Figure 7.9: Functional model of a single-stage Carrier Ethernet switch.

identified in most measurement studies. We base our discussion on the data shown in Figure 22 of [Job09]. The upper part of the figure shows PTD samples obtained with a step-wise change in background load every two hours, with the following load levels: 10%, 40%, 70%, 100%, 70%, 40%, 10%, 100%, 10%, 70%, 0%, corresponding to an average load of about 47% (Figure 19 in [Job09]).

The data of [Job09] clearly demonstrates that a background load above 0% results in increased p2pPDV, even though PTP packets have highest priority. This effect is caused by the non-preemptive operation mode of switches: If a PTP packet arrives while the switch is transmitting a background packet, processing of the PTP packet is delayed until the background packet has been sent. The delay experienced by the PTP packet is bounded from above by the maximum time required for transmitting a background packet.[3] With a constant data rate, this time depends only on the size of the largest possible background packet. The PDV is then a mixture of the point mass at zero (no background packet was being processed) and the distribution of background packet transmission times. In practice, these interactions result in a characteristic, load-dependent shape for the density of the PDV distribution. The effect of load is visible in Figure 12 of [Cos08], and in Figure 8 of [BGNV09] where the density of the PTD distribution is shown: For low load levels, the density of the PDV distribution is close to the point mass at zero (recall that the PDV distribution is the shifted PTD distribution, i.e. any constant offset in the PTD distribution can be ignored). The higher the load, the more likely it becomes that PTP packets must wait for a background packet, and therefore the distribution of background packet

[3]Cf. [BDL09], where this is referred to as the 'jumbo packet phenomenon'.

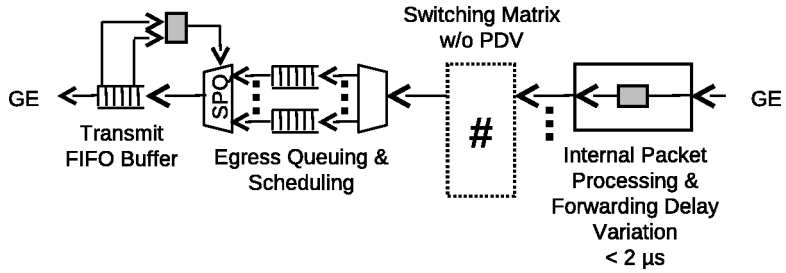


Figure 7.10: Delay model of a CE switch.

transmission times becomes more dominant. Note that the shape of the PDV density at 50% in [Cos08] is very similar to those in Figure 5 of [HSXX] (50% load) and Figure 22 of [Job09] (about 47% load). There is a peak at low values, and a drawn-out block reminiscient of the density of the uniform distribution. The first requirement on simulation models is thus to be able to reproduce this characteristic, load-dependent shape of the PDV density.

Considering again the PTD data shown in Figure 22 of [Job09], we observe that at a background load of 100% both PTD_{min} and PTD_{max} increase drastically, giving the plot the appearence of being 'shifted up'. The same 'delay step' at 100% load can be observed in Figures 5 and 7 of [DHK+01], Figures 11, 14, and 15 of [AWGG04], and in [ZHB10]. This delay step is due to the internal layout of typical switches. As discussed in the next section, packets leaving the switch must pass a transmit FIFO buffer before being transmitted. This buffer, however, does not support priorities. If the transmit FIFO buffer runs into overload, it throttles the egress scheduler in order to reduce the load. The additional delay affects all packets, including the high-priority PTP packets. Our second requirement on simulation models is thus that they are able to represent the delay step at overload.

Note that the increased delay in overload situations does not constitute a problem in itself, since a constant delay offset can be ignored by the slave clocks. However, overload situations occur only intermittently. Then, the resulting delay steps increase overall PDV, and thus affect PTP syntonisation accuracy.

7.3.3 Simulation Models for PDV Evaluation

The need for simulation models is dictated by the limited coverage of measurement studies on practical networking equipment. Even though there exist guidelines for conducting measurement studies [ITU08], cost and time constraints render exhaustive tests infeasible. Due to this restriction, network operators cannot evaluate and compare different backhaul network design choices with respect to their suitability for PTP. Discrete-event simulation allows evaluation in a much more efficient way

than black-box testing. However, the high accuracy requirements of PTP demand very precise models, because the delay variation experienced by the fastest packets when passing through a node must be quantifiable with microsecond precision under a wide range of load and other conditions. As these requirements are far beyond those of typical applications, and product documentation does not provide the necessary detail, sufficiently accurate models for networking equipment are not available in state-of-the-art network simulators. In particular, current models do not include many of the internal structures that have an effect on the PDV.

In [WRM11] we showed that the PDV behaviour described in the previous section requires highly-detailed models of the switching hardware. In turn, such detailed models require detailed insight into the internal structure of the switch. In fact, the level of detail needed is usually not reflected in manufacturer's data sheets. Based on a careful investigation of the functional structure of a typical single-stage Carrier Ethernet switch we arrived at the detailed model shown in Figure 7.10 The model contains a transmit FIFO buffer and an arbitrary delay element, neither of which are documented in typical manufacturer data sheets. A simulation model of the switch that only took into account the documented features of the switch would fail to represent their effects, as we showed in [WRM11].

Given a detailed model, the PDV of a backhaul network can be evaluated by connecting several copies of the detailed switch model to form a chain between the master and slave clocks. We then need to generate background traffic on the intermediate switches and transmit PTP packets through this chain, measuring the delay for the PTP packets. We use the state-of-the art discrete-event simulator ns-2 [ns2] for the simulation. In order to generate background load, traffic consisting of individual packets was generated in addition to the PTP traffic used for the measurements. Due to the high data rates in state-of-the-art backhaul networks, discrete-event simulation of high loads and long network paths entails a large number of background packets, and thus a very large number of events, which in turn increase simulation times. Consequently, the detailed simulation approach does not scale well to situations where PTP packets must traverse long paths.

7.3.4 Hybrid Simulation for PDV Evaluation

For scenarios where background packets can enter and leave the path at any intermediate switch, the problem of long simulation times can be addressed by a hybrid approach to simulation where phase-type distributions are used: First, PDV is measured on a highly-detailed model of a single switch. These measurements are then approximated by a PH distribution. Since we are interested in the 1% quantile of the PDV of the simulated system, fitting the 1% quantile well was particularly important. We used the PhFit tool [HT02] to fit the data set with an acyclic phase-type distribution of size 20. The detailed switch model and the detailed background-packet simulation were then replaced by a simple server that draws delays for PTP

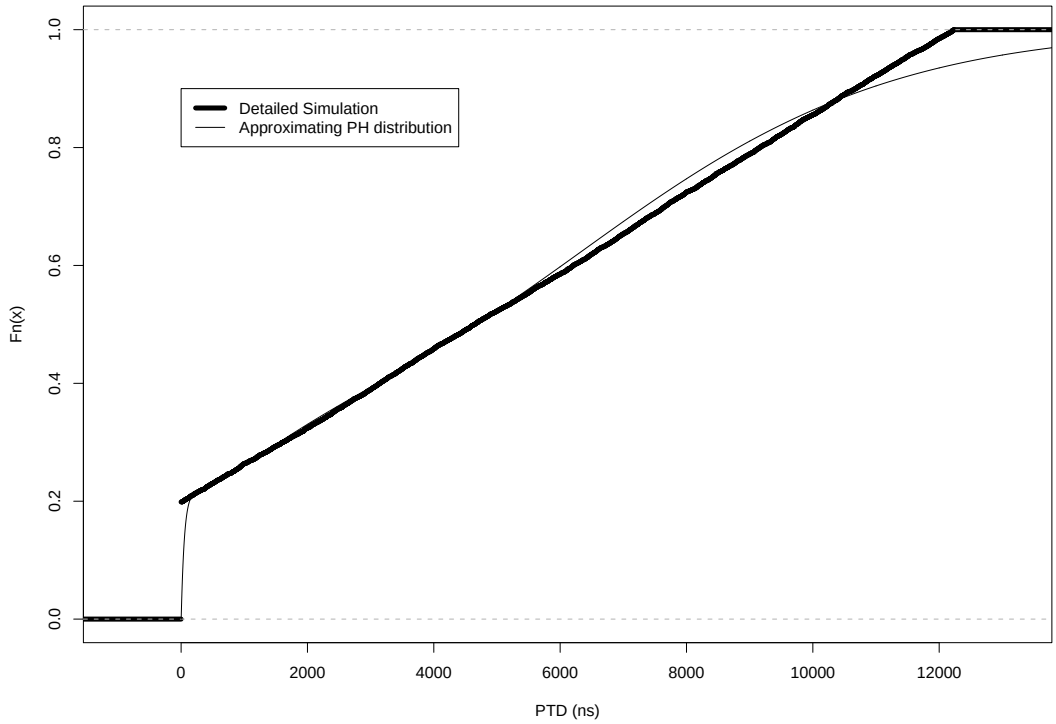


Figure 7.11: CDF of PTD at 80% background load with detailed simulation and PH approximation using 1 switch.

packets from the fitted PH distribution, using the libphprng library. In this simpler model only PTP packets and their delays need to be simulated, and thus only a very small number of events has to be processed.

As discussed in [WRM11], the detailed simulation models capture the behaviour of typical networking hardware well. In order to illustrate the validity of the PH approximations we simulated scenarios with 1 link and with 20 links, using both the PH approximation and the detailed models. The Cumulative Density Functions (CDFs) of the resulting PDV distributions are shown in Figures 7.11 and 7.12. Note that the CDFs from the simulations using the approximations fit the lower quantiles of the CDFs from the detailed simulations well, but tend to diverge on the higher quantiles. This is due to the fact that PH distributions have infinite support, in contrast to the limited support of the PDV distribution. However, as PTP slave clocks only use data from the lower quantiles of the PDV distribution, this inaccuracy is negligible in PTP simulations. We investigate the error of the 1% quantile of the PDV in more detail in Figure 7.13(a). Note that both the relative and the absolute error stay well below 1 μs, even for a large number of links.

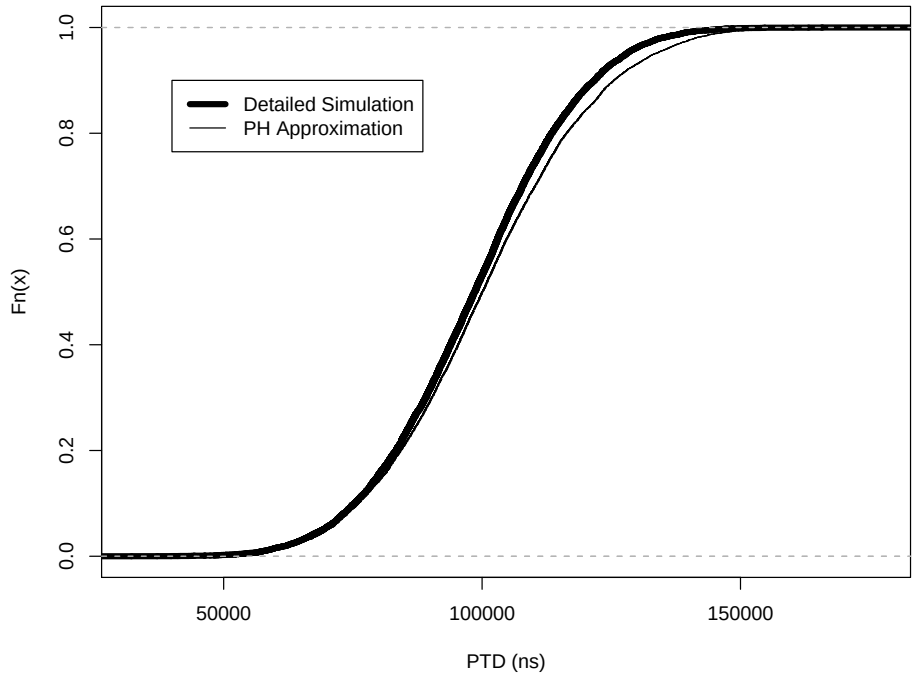


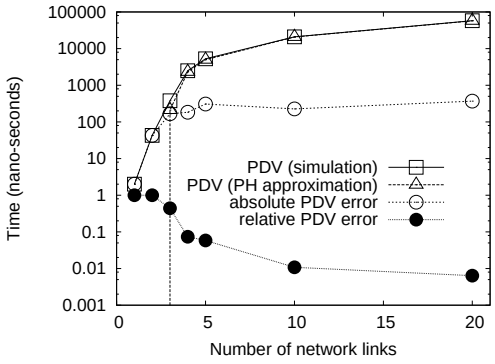
Figure 7.12: CDF of PTD at 80% background load with detailed simulation and PH approximation, using 20 switches.

The advantage of using PH approximations is illustrated in Figure 7.13(b), where we show the simulation times for increasing numbers of links. Observe that the time required for the detailed simulation rises quickly, while the time for the approximated simulations stays in the order of minutes.

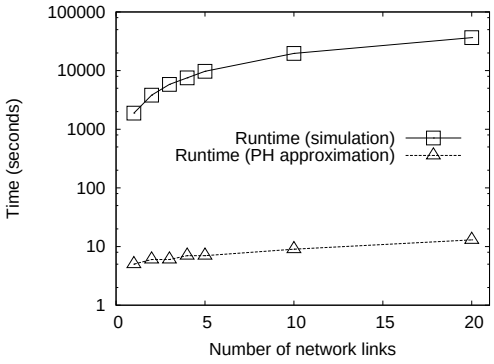
7.4 Concluding Remarks

In this chapter we introduced the libphprng library for random-variate generation from phase-type distributions and presented two case-studies illustrating the use of phase-type distributions in system evaluation.

The first case-study demonstrated the advantage in accuracy that phase-type models provide over less flexible methods of fitting data. The case-study was complemented by a performance evaluation of the costs of random-variate generation in simulation models, which showed that random variates from PH distributions require significantly more CPU time than more basic models, like the exponential and lognormal distributions. However, methods that exploit special structures reduce the computational overhead to a reasonable level.



(a) Error in the 1% quantile of the PTD distribution for PH approximation (logarithmic error axis).



(b) Simulation times with detailed simulation and with PH approximation (logarithmic time axis).

Figure 7.13: Comparison of detailed simulation and PH approximation.

The second case-study illustrated how phase-type distributions can be used in a hybrid approach to system evaluation using discrete-event simulation. By approximating the behaviour of components of our simulation model using PH distributions we could drastically reduce simulation run-times and thus improve scalability enormously. The performance gain by complexity reduction is indeed impressive, as detailed simulation requires several days, while the simulation with PH distribution finishes within minutes. At the same time, we could show that the abstraction did not sacrifice accuracy of the results.

Conclusion

In this work we studied methods for efficient and accurate system evaluation. In the first part of the thesis we took a broad perspective on important requirements for system-evaluation studies. We considered stochastic fault-models as tools to describe and reproduce faults in system evaluation. Such models are a prerequisite of studying quality-of-service impairments caused by faults as well as methods to counteract these effects. The survey of existing stochastic fault-models we presented here employs a hierarchical approach to sub-divide complex service-oriented systems such that relevant fault-models may be identified. With its broad overview of fault-models, this survey enables researchers to parameterise their evaluation studies using real-world fault-models. While the survey itself focusses on service-oriented systems, the collected fault-models can benefit a wide range of complex distributed systems. In a case-study we considered the effects of a specific class of fault-models for IP packet loss emulation and illustrated that fault-models that represent the behaviour of a fault well are not necessarily suited for reproducing the fault in an evaluation study. We showed that our implementation of a more appropriate model gives realistic results.

We then investigated phase-type distributions as one important class of fault-models. Starting with a discussion of the theoretical basics, we focussed on the application of these distributions in simulation-based and experimental approaches. The algorithms for random-variate generation and the optimisation methods for reducing the computational costs of random-variate generation developed here enable an efficient way of using highly-accurate representations of real-world distributions in system evaluation. A user-friendly cluster-based fitting approach and an easy-to-use dynamic library for random-variate generation from PH distributions complement these results and enable non-experts to benefit from accurate and efficient models. Two case-studies highlight the advantages of using PH distributions in simulation-based evaluation.

The work presented here opens several interesting directions for further research. First, we may consider extension of the theoretical work on optimal representations of PH distributions for random-variate generation. Our results hold for the most important canonical representations for the APH and the general PH class, and

are thus valid for the whole of the APH and PH classes, respectively. As we have observed in our discussion of the costs of the algorithms, sub-classes can give representations and algorithms that reduce costs further. The search for optimality results for these sub-classes might be difficult, but appears potentially rewarding.

A second important area of future work is the application of our results for phase-type distributions in random-variate generation from Markovian arrival processes. As we have noted in our evaluation study, existing algorithms for MAP random-variate generation are generally much slower than algorithms limited to PH. Since MAPs are a generalisation of PH distributions, it may be possible to improve efficiency by applying the methods and results for the PH class.

Third, as we illustrated in the case-study on PTP evaluation, phase-type distributions can help to improve scalability of complex simulation models tremendously. In this case-study we modelled complex parts of a simulation model by fitting PH distributions, and, using our efficient algorithms, we could then simulate the model in a fraction of the original time. Generalising and automating this approach is certainly an enticing avenue for further research.

Finally, the survey in the first part of the thesis identified several areas that are not yet well represented by fault-models based on empirical studies. Detailed measurement studies and rigorous fitting of stochastic models to the results will provide models that may help future researchers understand the nature of QoS impairments in real systems. This understanding will facilitate the development of methods to address these issues in an effective manner.

Appendices

Appendix A

Small Proofs

Proof for Lemma 2.3.1.

$$\begin{aligned}
\left(\mathbf{S}^{-1}\mathbf{Q}\mathbf{S}\right)^k &= \left(\mathbf{S}^{-1}\mathbf{Q}\mathbf{S}\right)\left(\mathbf{S}^{-1}\mathbf{Q}\mathbf{S}\right)\left(\mathbf{S}^{-1}\mathbf{Q}\mathbf{S}\right)^{k-2} && \text{(A.1)}\\
&= \left(\mathbf{S}^{-1}\mathbf{Q}^2\mathbf{S}\right)\left(\mathbf{S}^{-1}\mathbf{Q}\mathbf{S}\right)^{k-2} && \text{(A.2)}\\
&= \cdots = \mathbf{S}^{-1}\mathbf{Q}^k\mathbf{S}, && \text{(A.3)}
\end{aligned}$$

and, using the series expansion of $\mathrm{e}^{\mathbf{Q}}$ and (2.20):

$$\begin{aligned}
\mathrm{e}^{\mathbf{S}^{-1}\mathbf{Q}\mathbf{S}} &= \sum_{l=0}^{\infty}\frac{1}{l!}\left(\mathbf{S}^{-1}\mathbf{Q}\mathbf{S}\right)^l && \text{(A.4)}\\
&= \sum_{l=0}^{\infty}\frac{1}{l!}\mathbf{S}^{-1}\mathbf{Q}^l\mathbf{S} && \text{(A.5)}\\
&= \mathbf{S}^{-1}\left(\sum_{l=0}^{\infty}\frac{1}{l!}\mathbf{Q}^l\right)\mathbf{S} && \text{(A.6)}\\
&= \mathbf{S}^{-1}\,\mathrm{e}^{\mathbf{Q}}\,\mathbf{S} && \text{(A.7)}
\end{aligned}$$

□

Proof for Lemma 2.3.2. $\left(\boldsymbol{\alpha}\mathbf{S}, \mathbf{S}^{-1}\mathbf{Q}\mathbf{S}\right)$ defines the distribution

$$\begin{aligned}
G(t) &= 1 - \boldsymbol{\alpha}\mathbf{S}\,\mathrm{e}^{\mathbf{S}^{-1}\mathbf{Q}\mathbf{S}t}\,\mathbb{1} = 1 - \boldsymbol{\alpha}\mathbf{S}\mathbf{S}^{-1}\,\mathrm{e}^{\mathbf{Q}t}\,\mathbf{S}\mathbb{1} = 1 - \boldsymbol{\alpha}\,\mathrm{e}^{\mathbf{Q}t}\,\mathbb{1} && \text{(A.8)}\\
&= F(t). && \text{(A.9)}
\end{aligned}$$

□

Proof for Lemma 2.3.3. Consider the distribution defined by $(\boldsymbol{\alpha}\hat{\mathbf{W}}, \mathbf{Q}')$:

$$
\begin{aligned}
G(t) &= 1 - \boldsymbol{\alpha}\hat{\mathbf{W}}\,\mathrm{e}^{\mathbf{Q}'t}\,\mathbb{1}\!\mathbb{I} = 1 - \boldsymbol{\alpha}\hat{\mathbf{W}}\left(\sum_{l=0}^{\infty}\frac{1}{l!}(\mathbf{Q}'t)^l\right)\mathbb{1}\!\mathbb{I} && \text{(A.10)}\\
&= 1 - \boldsymbol{\alpha}\hat{\mathbf{W}}\left(\mathbf{I} + \mathbf{Q}'t + \frac{(\mathbf{Q}'t)^2}{2} + \dots\right)\mathbb{1}\!\mathbb{I} && \text{(A.11)}\\
&= 1 - \boldsymbol{\alpha}\left(\hat{\mathbf{W}}\mathbf{I} + \hat{\mathbf{W}}\mathbf{Q}'t + \hat{\mathbf{W}}\frac{(\mathbf{Q}'t)^2}{2} + \dots\right)\mathbb{1}\!\mathbb{I} && \text{(A.12)}\\
&= 1 - \boldsymbol{\alpha}\left(\mathbf{I}\hat{\mathbf{W}} + \mathbf{Q}\hat{\mathbf{W}}t + \mathbf{Q}\hat{\mathbf{W}}\frac{\mathbf{Q}'t^2}{2} + \dots\right)\mathbb{1}\!\mathbb{I} && \text{(A.13)}\\
&= 1 - \boldsymbol{\alpha}\left(\mathbf{I}\hat{\mathbf{W}} + \mathbf{Q}\hat{\mathbf{W}}t + \mathbf{Q}\mathbf{Q}\hat{\mathbf{W}}\frac{t^2}{2} + \dots\right)\mathbb{1}\!\mathbb{I} && \text{(A.14)}\\
&= 1 - \boldsymbol{\alpha}\left(\sum_{l=0}^{\infty}\frac{1}{l!}(\mathbf{Q}t)^l\hat{\mathbf{W}}\right)\mathbb{1}\!\mathbb{I} && \text{(A.15)}\\
&= 1 - \boldsymbol{\alpha}\,\mathrm{e}^{\mathbf{Q}t}\,\mathbb{1}\!\mathbb{I} = F(t). && \text{(A.16)}
\end{aligned}
$$

□

Appendix B

Stochastic Modelling Miscellany

B.1 Probability Distributions

Several probability distributions are used in the context of this work. We summarise their properties here.

The Exponential Distribution The exponential distribution is the simplest phase-type distribution. It has length $n = 1$ and the rate $\lambda > 0$ as its only parameter. The probability density function (PDF), cumulative distribution function (CDF), mean, variance, and squared coefficient of variation cv^2 are, respectively:

$$\begin{aligned} f(t) &= \lambda \mathrm{e}^{-\lambda t} && \text{(B.1)} \\ F(t) &= 1 - \mathrm{e}^{-\lambda t} && \text{(B.2)} \\ Mean &= \frac{1}{\lambda} && \text{(B.3)} \\ Variance &= \lambda^2 && \text{(B.4)} \\ cv^2 &= \frac{\lambda^2}{\lambda^2} = 1. && \text{(B.5)} \end{aligned}$$

Using the inversion method, a random variate t from the exponential distribution with rate λ can be drawn using a uniform random number u as [Jai91]:

$$\begin{aligned} u &= F(t) = 1 - \mathrm{e}^{-\lambda t} && \text{(B.6)} \\ 1 - u &= \mathrm{e}^{-\lambda t} && \text{(B.7)} \\ \ln(1-u) &= -\lambda t && \text{(B.8)} \\ t &= -\frac{\ln(1-u)}{\lambda t} && \text{(B.9)} \\ &\sim -\frac{\ln u}{\lambda_t}. && \text{(B.10)} \end{aligned}$$

The simplification from $1 - u$ to u in the last step is possible because $1 - u$ and u are both uniform random numbers on $(0, 1)$ [Hav98].

The Erlang Distribution The Erlang distribution with rate $\lambda > 0$ and length $k \in \mathbb{N}^+$ is the sum of k independent random variables with exponential distribution with rate λ. It has probability density function (PDF), cumulative distribution function (CDF), mean, variance, and squared coefficient of variation cv^2 as follows:

$$f(t) = \frac{\lambda^k t^{k-1} \mathrm{e}^{-\lambda t}}{(k-1)!} \tag{B.11}$$

$$F(t) = 1 - \mathrm{e}^{-\lambda t} \tag{B.12}$$

$$Mean = \frac{k}{\lambda} \tag{B.13}$$

$$Variance = \frac{k}{\lambda^2} \tag{B.14}$$

$$cv^2 = \frac{k}{\lambda^2}\frac{\lambda^2}{k^2} = \frac{1}{k}. \tag{B.15}$$

The Geometric Distribution The geometric distribution with parameter $p \in [0, 1]$ can be defined in two different ways. It can describe the number of Bernoulli trials with success probability p required to get one success or the number of failures of successive Bernoulli trials before the first success. If the geometric distribution describes the number of trials required to get one success, its support is $\mathbb{N}^+ = \{1, 2, \dots\}$ and it has the following probability mass function (PMF), cumulative distribution function, and mean:

$$Pr(X = n) = (1 - p)^{n-1} p \tag{B.16}$$

$$Pr(X \leq n) = 1 - (1 - p)^n \tag{B.17}$$

$$Mean = \frac{1}{p}. \tag{B.18}$$

A geometric distribution that describes the number of failures before the first success has support $\mathbb{N} = \{0, 1, 2, \dots\}$ and PMF, CDF, and mean as follows:

$$Pr(X = n) = (1 - p)^n p \tag{B.19}$$

$$Pr(X \leq n) = 1 - (1 - p)^{n+1} \tag{B.20}$$

$$Mean = \frac{1 - p}{p}. \tag{B.21}$$

Given a uniform random number on $(0, 1)$, a random variate from the geometric distribution with success probability p and support $\mathbb{N}^+$ can be obtained using the

inversion method [Jai91, Law06]:

$$
\begin{aligned}
u &= 1-(1-p)^n && \text{(B.22)}\\
1-u &= (1-p)^n && \text{(B.23)}\\
n &= \left\lceil \frac{\ln(1-u)}{\ln(1-p)} \right\rceil && \text{(B.24)}\\
&\sim \left\lceil \frac{\ln(u)}{\ln(1-p)} \right\rceil && \text{(B.25)}\\
& && \text{(B.26)}
\end{aligned}
$$

Likewise, a random variate from the geometric distribution with support $\mathbb{N}$ can be generated by

$$
n = \left\lfloor \frac{\ln(u)}{\ln(1-p)} \right\rfloor . \qquad \text{(B.27)}
$$

B.2 The Embedded Markov Chain (EMC)

Every Markovian representation $(\boldsymbol{\alpha}, \mathbf{Q})$ of a phase-type distribution can be interpreted as a continuous-time Markov chain that is entered at some state i and then traversed until the absorbing state is reached. The PH distribution is then the distribution of the sum of sojourn times for all the paths taken through the CTMC. The matrix $\mathbf{Q}$ describes both the sojourn times and the possible state transitions in the transient part. The tuple $((\boldsymbol{\alpha}, 0), \hat{\mathbf{Q}})$ is the complete CTMC, including the absorbing state.

The Embedded Markov Chain (EMC) comes in handy if we are only interested in the state transitions. The EMC has the same initialisation vector $(\boldsymbol{\alpha}, 0)$, but is a discrete-time Markov chain (DTMC) and therefore has a matrix of transition probabilities, rather than rates. The following derivation roughly follows [Ste09], p. 20: In each row i of $\hat{\mathbf{Q}}$ the entries $\lambda_{ij}, j = 1, \ldots, n+1; j \neq i$ give the rates of exponential distributions that describe the time until transition to the jth state, while λ_{ii} is the rate of the exponential distribution for the sojourn time of the ith state. The probability of entering the jth state depends on the rate with which j is entered, i.e. states with a high rate have a higher probability of being the next state than states with a low rate. Thus, for $j \neq i$,

$$
p_{ij} = \frac{\lambda_{ij}}{\lambda_{ii}} \qquad \text{(B.28)}
$$

gives the probability of entering state j from state i. Furthermore, the CTMC

cannot immediately enter the same state again, i.e.

$$p_{ii} = 0. \tag{B.29}$$

The transition matrix for the EMC is therefore

$$\mathbf{P} = \mathbf{I} - \mathrm{Diag}(\hat{\mathbf{Q}})^{-1}\hat{\mathbf{Q}}, \tag{B.30}$$

where $\mathrm{Diag}(\hat{\mathbf{Q}})$ is the diagonal of the matrix $\hat{\mathbf{Q}}$.

Appendix C

Moment-Matching for Erlang Distributions

We use the following efficient method for fitting the first two moments of an Erlang distribution to a data set. The method has been described in [Kra12].

Let m and cv^2 denote the mean and squared coefficient of variation (SCV) of the data set, respectively. We first estimate the length of the Erlang distribution based on the SCV. Recall that for the Erlang distribution

$$cv^2 = \frac{1}{k},$$

and thus

$$k = \frac{1}{cv^2} \tag{C.1}$$

gives the number of phases required. The mean of the Erlang distribution with length k and rate λ is

$$m = \frac{k}{\lambda},$$

and therefore

$$\lambda = \frac{k}{m} \tag{C.2}$$

can be used to compute the rate parameter λ.

If k in (C.1) is a positive natural number, then we use (C.2) to compute the rate

λ. The final parameter set is then $(\hat{k}, \hat{\lambda})$ with

$$\hat{k} = \frac{1}{cv^2} \tag{C.3}$$

$$\hat{\lambda} = \frac{k}{m}. \tag{C.4}$$

If $k \notin \mathbb{N}^+$, we compute the two closest k that fulfill this constraint:

$$k_1 := \min\left\{1, \left\lfloor \frac{1}{cv^2} \right\rfloor\right\} \tag{C.5}$$

$$k_2 := k_1 + 1. \tag{C.6}$$

Using (C.2), we obtain two distributions $F_{k_1,\lambda_1}, F_{k_2,\lambda_2}$ that match the mean of the data set:

$$\lambda_1 := \frac{k_1}{m} \tag{C.7}$$

$$\lambda_2 := \frac{k_2}{m}. \tag{C.8}$$

We then select the parameter set $(\hat{k}, \hat{\lambda})$ such that the log-likelihood of the fitted distribution is maximised:

$$(\hat{k}, \hat{\lambda}) := \begin{cases} (k_1, \lambda_1) & : \mathcal{L}(F_{k_1,\lambda_1}) > \mathcal{L}(F_{k_2,\lambda_2}) \\ (k_2, \lambda_2) & : \text{else} \end{cases}. \tag{C.9}$$

Appendix D

Projection Methods

The MonoFit algorithm employs projection into the feasible region or onto its boundaries to ensure that the constraints of the optimisation problem are fulfifilled. In this appendix we give details on the projection methods we used in the algorithm.

Constraint (5.11) requires that the sum of the entries of the initial vector must be 1. This is ensured by setting the initial vector to

$$\boldsymbol{\alpha} \quad := \quad \frac{\boldsymbol{\alpha}}{\boldsymbol{\alpha}\mathbb{1}}, \tag{D.1}$$

i.e. projecting the vector onto the region defined by $\boldsymbol{\alpha}\mathbb{1} = 1$.

Constraints (5.12) and (5.14) state that initial probabilities and feedback probabilities must not be negative. We ensure this by mapping negative values onto 0, which is the smallest value that satisfies the constraints:

$$\alpha_i \quad := \quad \max(0, \alpha_i) \text{ for } i = 1, \ldots, n, \tag{D.2}$$

$$z_i \quad := \quad \max(0, z_i) \text{ for } i = 1, \ldots, m. \tag{D.3}$$

Constraint (5.16) requires that all block lengths b_i are natural numbers greater than zero. This constraint is ensured by rounding all values to the nearest integer larger than or equal to one:

$$b_i \quad := \quad \max(1, \texttt{Round}(b_i)) \text{ for } i = 1, \ldots, m. \tag{D.4}$$

Finally, (5.17) demands that the size of the distribution stays constant and equal to n. This constraint may be violated not only by computations of new vertices during the course of the algorithm, but also by projection to ensure (5.16). Although the following approach lacks a strong theoretical foundation, it has proved to yield

good results in our experiments: Let n' be the sum of the block lengths:

$$n' := \sum_{i=1}^{m} b_i. \tag{D.5}$$

If $n' < n$, we add the missing phases to the first block:

$$b_1 := b_1 + n - n'. \tag{D.6}$$

If $n' > n$, then we remove as many phases as possible from each block (while ensuring that the block size is at least 1), starting with the first, until both lengths are equal. The following pseudo-code illustrates the method:

$n' := \sum_{i=1}^{m} b_i$
$\delta := n' - n$
for $i = 1, \ldots, m$ **do**
  $b'_i := b_i$
  $b_i := \max(1, b'_i - \delta)$
  $\delta := \delta - (b_i - b'_i)$
end for

Zusammenfassung

Die Evaluation komplexer Systeme erfordert effiziente und genaue Untersuchungsmethoden. Abhängig vom Abstraktionsniveau existieren unterschiedliche Verfahren, die Messungen in Testbetten ebenso umfassen wie Simulationen und analytische Ansätze. Die Verfahren unterscheiden sich hinsichlich ihrer Effizienz und der Genauigkeit der erzielbaren Ergebnisse. Für realistische Aussagen sollten in einer Evaluation immer Methoden aller Abstraktionsebenen kombiniert werden. Überdies sollten auf allen Ebenen Phänomene der echten Welt in die Untersuchung einfließen. Dies erfordert den Einsatz stochastischer Modelle für die Fehler, die das System beeinträchtigen können. In dieser Arbeit werden wichtige Aspekte der Systemevaluation mit stochastischen Fehlermodellen untersucht.

Im ersten Teil der Arbeit wird eine Übersicht über stochastische Fehlermodelle erarbeitet und ein neues Klassifikationsschema für Fehler in Service-Orientierten Systemen vorgestellt.

Der zweite Teil der Arbeit konzentriert sich auf den Einsatz von Phasentypverteilungen als Fehlermodelle. Es werden Algorithmen zur Zufallszahlenerzeugung aus Phasentypverteilungen und ihre Kosten untersucht. Anschließend wird ein Verfahren vorgestellt, das Phasentypverteilungen so umformt, daß die Kosten der Zufallszahlenerzeugung minimiert werden. Für diesen Ansatz wird ein Optimalitätsresultat für die Unterklasse der azyklischen Phasentypverteilungen hergeleitet. Des weiteren wird ein auf Clustering basierender Algorithmus zur effizienten und nutzerfreundlichen Anpassung von Phasentypverteilungen an Datensätze entwickelt. Der Algorithmus wurde im Programm HyperStar implementiert. HyperStar kann überdies als graphische Benutzeroberfläche für die Entwicklung von Fitting-Algorithmen dienen. Diese Verwendungsweise wird anhand eines neuen Algorithmus' zur Anpassung allgemeiner Phasentypverteilungen demonstriert.

Im dritten Teil werden die Ergebnisse des ersten und zweiten Teils in der Anwendung illustriert. Es wird zunächst die Anwendung stochastischer Fehlermodelle für IP-Paketverluste in der Fehlerinjektion untersucht. Der Schwerpunkt liegt hierbei auf der Wechselwirkung zwischen Fehlermodellen und Verkehrsankunftsströmen und ihrem Einfluß auf die Evaluationsergebnisse. Anschließend wird eine Bibliothek zur effizienten Zufallszahlenerzeugung in Simulationen vorgestellt und in einer Fallstudie evaluiert. Eine zweite Fallstudie demonstriert die mögliche Effizienzsteigerung durch den Einsatz von Phasentypverteilungen in Simulationen.

Ich versichere, daß ich alle Quellen und Hilfsmittel angegeben und auf deren Grundlage die Arbeit selbständig verfaßt habe. Die Arbeit wurde nicht bei einem früheren Promotionsverfahren eingereicht.

Bibliography

[ALRL04] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[AMM10] Raquel Almeida, Naaliel Mendes, and Henrique Madeira. Sharing experimental and field data: The amber raw data repository experience. In *Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems Workshops*, ICDCSW '10, pages 313–320, Washington, DC, USA, 2010. IEEE Computer Society.

[ANO96] S. Asmussen, O. Nerman, and M. Olsson. Fitting Phase-Type Distribution Via the EM Algorithm. *Scand. J. Statist.*, 23:419–441, 1996.

[AS87] D. Aldous and L. Shepp. The least variable phase-type distribution is erlang. *Stochastic Models*, 3:467–473, 1987.

[AWGG04] A.Jacobs, J. Wernicke, B. Gordon, and A. George. Characterization of quality of service in commercial ethernet switches for statistically bounded latency in aircraft networks. Technical report, Highperformance Computing and Simulation (HCS) Research Lab, Department of Electrical and Computer Engineering, University of Florida, 2004. Available online http://www.hcs.ufl.edu/~jacobs/rockwell_qos.doc (Last seen 14 February 2011).

[BBH+11] Levente Bodrog, Peter Buchholz, Armin Heindl, András Horváth, Gábor Horváth, István Kolossváry, Zoltán Németh, Philipp Reinecke, Miklós Telek, and Miklós Vécsei. Butools: Program packages for computations with PH, ME distributions and MAP, RAP processes. http://webspn.hit.bme.hu/~butools, October 2011.

[BBK10] Falko Bause, Peter Buchholz, and Jan Kriege. ProFiDo - The Processes Fitting Toolkit Dortmund. In *Proc. of the 7th International Conference on Quantitative Evaluation of SysTems (QEST 2010)*, pages 87–96. IEEE Computer Society, 2010.

[BCH+12] Jeremy T. Bradley, Lucia Cloth, Richard Hayden, Leïla Kloul, Philipp Reinecke, Markus Siegle, Nigel Thomas, and Katinka Wolter. Scalable Stochastic Modelling for Resilience. In Katinka Wolter, Alberto Avritzer, Marco Vieira, and Aad van Moorsel, editors, *Resilience Assessment and Evaluation of Computing Systems*, chapter 6, pages 115–149. Springer, 2012.

[BDL09] Dinh Thai Bui, Arnaud Dupas, and Michel Le Pallec. Packet delay variation management for a better ieee1588v2 performance. In *International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pages 1–6, Brescia, Oct. 2009.

[Ber09] D.S. Bernstein. *Matrix Mathematics: Theory, Facts, and Formulas*. Princeton University Press, 2 edition, 2009.

[BGJ+11] Bastian Blywis, Mesut Günes, Felix Juraschek, Oliver Hahm, and Nicolai Schmittberger. Properties and Topology of the DES-Testbed (2nd Extended Revision). Technical Report TR-B-11-04, Freie Universität Berlin, July 2011.

[BGNV09] Jeff Burch, Kenneth Green, John Nakulski, and Dieter Vook. Verifying the performance of transparent clocks in ptp systems. In *Precision Clock Synchronization for Measurement, Control and Communication, 2009. ISPCS 2009. International Symposium on*, pages 1 –6, October 2009.

[BGPS07] Lakshmi N. Bairavasundaram, Garth R. Goodson, Shankar Pasupathy, and Jiri Schindler. An Analysis of Latent Sector Errors in Disk Drives. In *SIGMETRICS 2007*, pages 289–300, New York, NY, USA, 2007. ACM.

[BHT05] A. Bobbio, A. Horváth, and M. Telek. Matching three moments with minimal acyclic phase type distributions. *Stochastic Models*, 21(2):303–326, 2005.

[BIMT05] BEA Systems, IBM, Microsoft Corporation Inc, and TIBCO Software, Inc. Web Services Reliable Messaging Protocol (WS-ReliableMessaging), February 2005.

[BLS+09] David Bernstein, Erik Ludvigson, Krishna Sankar, Steve Diamond, and Monique Morrow. Blueprint for the Intercloud - Protocols and Formats for Cloud Computing Interoperability. In *ICIW '09: Proceedings of the*

2009 Fourth International Conference on Internet and Web Applications and Services, pages 328–336, Washington, DC, USA, 2009. IEEE Computer Society.

[Box65] M. J. Box. A new method of constrained optimization and a comparison with other methods. *The Computer Journal*, 8(1):42–52, 1965.

[BPdL98] E. Brown, J. Place, and A. Van de Liefvoort. Generating Matrix Exponential Random Variates. *Simulation*, 70:224–230, April 1998.

[BRC09] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N. Calheiros. Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities. In *Proceedings of the 7th High Performance Computing and Simulation (HPCS 2009) Conference*, volume abs/0907.4878, Leipzig, Germany, June 21 – 24 2009.

[BS03] Paul Barford and Joel Sommers. A Comparison of Probe-based and Router-based Methods for Measuring Packet Loss. Technical report, University of Wisconsin-Madison, September 2003.

[BS04] Paul Barford and Joel Sommers. Comparing Probe- and Router-Based Packet-Loss Measurement. *IEEE Internet Computing*, 8(5):50–56, 2004.

[BT94] A. Bobbio and M. Telek. A Benchmark for PH Estimation Algorithm: Results for Acyclic-PH. *Stochastic Models*, 10:661–677, 1994.

[BWM07a] Stefan Brüning, Stephan Weißleder, and Miroslaw Malek. A Fault Taxonomy for Service-Oriented Architecture. Technical Report 215, Humboldt-Universität zu Berlin, 2007. [Online: Stand 2008-10-17T07:41:39Z].

[BWM07b] Stefan Brüning, Stephan Weißleder, and Miroslaw Malek. A Fault Taxonomy for Service-Oriented Architecture. In *IEEE International Symposium on High-Assurance Systems Engineering*, pages 367–368, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[BYV+09] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25(6):599–616, 2009.

[CBS+07] K. S. May Chan, Judith Bishop, Johan Steyn, Luciano Baresi, and Sam Guinea. A Fault Taxonomy for Web Service Composition. In *Proceedings of the 3rd International Workshop on Engineering Service Oriented Applications (WESOA'07), Springer LNCS*, 2007.

[CFR03] D. Cotroneo, C.Di Flora, and S. Russo. Improving Dependability of Service Oriented Architectures for Pervasive Computing. In *Proceedings of the Eighth International Workshop on Object-Oriented Real-Time Dependable Systems, 2003 (WORDS 2003).*, pages 74–81, Los Alamitos, CA, USA, January 2003. IEEE Computer Society.

[Cos08] Lee Cosart. Studying network timing with precision packet delay measurements. In *Proc. 40th Annual Precise and Time Interval (PTTI) Meeting*, 2008.

[Cos09] Lee Cosart. Packet network timing measurement and analysis using an ieee 1588 probe and new metrics. In *Precision Clock Synchronization for Measurement, Control and Communication, 2009. ISPCS 2009. International Symposium on*, pages 1 –6, October 2009.

[Cri91] Flaviu Cristian. Understanding Fault-Tolerant Distributed Systems. *Communications of the ACM*, 34:56–78, 1991.

[CRM00] D. Costa, T. Rilho, and H. Madeira. Joint Evaluation of Performance and Robustness of a COTS DBMS Through Fault-Injection. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN) 2000*, pages 251–260, 2000.

[Cum82] A. Cumani. On the Canonical Representation of Homogeneous Markov Processes Modelling Failure-time Distributions. *Microelectronics and Reliability*, 22:583–602, 1982.

[CWA+05] Seung-Hwa Chung, Y. J. Won, D. Agrawal, Seong-Cheol Hong, Hong-Taek Ju, and Kihong Park. Detection and Analysis of Packet Loss on Underutilized Enterprise Network Links. In *E2EMON '05: Proceedings of the Workshop on End-to-End Monitoring Techniques and Services 2005*, pages 164–176, Washington, DC, USA, 2005. IEEE Computer Society.

[CZS08] Giuliano Casale, Eddy Z. Zhang, and Evgenia Smirni. Kpc-toolbox: Simple yet effective trace fitting using markovian arrival processes. In *Proceedings of the 2008 Fifth International Conference on Quantitative Evaluation of Systems*, pages 83–92, Washington, DC, USA, 2008. IEEE Computer Society.

[Dan10] Alexandra Danilkina. Empirischer Vergleich von Tools zur Approximation mit Phasentypverteilungen. Studienarbeit, Humboldt-Universität zu Berlin, May 2010. (in German).

[DC02] C. Demichelis and P. Chimento. IP Packet Delay Variation Metric for IP Performance Metrics (IPPM). RFC 3393 (Proposed Standard), November 2002.

[DCC+02] D. D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster. The Möbius Framework and Its Implementation. *Transactions on Software Engineering*, 28(10):956–969, 2002.

[DCGN03a] Michael Dahlin, Bharat Baddepudi V. Chandra, Lei Gao, and Amol Nayate. End-to-end WAN service availability. *IEEE/ACM Transactions on Networking*, 11(2):300–313, 2003.

[DCGN03b] Michael Dahlin, Bharat Baddepudi V. Chandra, Lei Gao, and Amol Nayate. End-to-end WAN service availability (Extended Version). Technical report, University of Texas at Austin, 2003.

[DGP05] V. Datla and K. Goseva-Popstojanova. Measurement-based Performance Analysis of e-Commerce Applications With Web Services Components. In *Proc. IEEE International Conference on e-Business Engineering ICEBE 2005*, pages 305–314, 2005.

[DHK+01] R. W. Dobinson, S. Haas, K. Korcyl, M. J. Levine, J. Lokier, B. Martin, C. Meirosu, F. Saka, and K. Vella. Testing and modeling ethernet switches and networks for use in atlas high-level triggers. *IEEE Trans. Nucl. Sci*, 48:607–612, 2001.

[Drä11] Matthias Dräger. Entwurf und Implementierung eines Moduls zur stochastischen Fehlerinjektion für IP-Netzwerke gemäß eines erweiterten Gilbert-Modells. Bachelor thesis, Freie Universität Berlin, 2011. (in German).

[ESH03] Rachid El Abdouni Khayari, Ramin Sadre, and Boudewijn R. Haverkort. Fitting world-wide web request traces with the em-algorithm. *Performance Evaluation*, 52(2-3):175–191, April 2003.

[Fad98] M.J Faddy. On inferring the number of phases in a coxian phase-type distribution. *Communications in Stochastic Models*, 14(1-2):407–417, 1998.

[FW56] Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.

[FW98] Anja Feldmann and Ward Whitt. Fitting mixtures of exponentials to long-tail distributions to analyze network. *Perform. Eval.*, 31(3-4):245–279, 1998.

[GHH03] Haifang Ge, Uli Harder, and Peter Harrison. Parameter Estimation for MMPPs using the EM Algorithm. In *UKPEW 2003*, 2003.

[GKT+08] A. Gorbenko, V. Kharchenko, O. Tarasyuk, Y. Chen, and A. Romanovsky. The Threat of Uncertainty in Service-Oriented Architecture. Technical Report 1122, Newcastle University, School of Computing Science, Oct 2008.

[GMKR07] A. Gorbenko, A. Mikhaylichenko, V. Kharchenko, and A. Romanovsky. Experimenting With Exception Handling Mechanisms Of Web Services Implemented Using Different Development Kits. Technical Report 1010, Newcastle University, School of Computing Science, March 2007.

[Hav98] B. R. Haverkort. *Performance of Computer Communication Systems: A Model-Based Approach*. John Wiley & Sons, Chichester, UK, 1998.

[HGHl08] Oliver Hohlfeld, Rüdiger Geib, and Gerhard Haß linger. Packet Loss in Real-Time Services: Markovian Models Generating QoE Impairments. In *Proc. of the 16th International Workshop on Quality of Service (IWQoS)*, pages 239–248, June 2008.

[HH08] Gerhard Haßlinger and Oliver Hohlfeld. The Gilbert-Elliott Model for Packet Loss in Real Time Services on the Internet. In Falko Bause and Peter Buchholz, editors, *MMB 2008*, pages 269–286. VDE Verlag, 2008.

[HNA02] T.J. Hacker, B.D. Noble, and B.D. Athey. The Effects of Systemic Packet Loss on Aggregate TCP Flows. In *Proceedings of the ACM/IEEE 2002 Conference on Supercomputing*, pages 1–15, 2002.

[HRTW12] Gábor Horváth, Philipp Reinecke, Miklós Telek, and Katinka Wolter. Efficient Generation of PH-distributed Random Variates. In Khalid Al-Begain, Dieter Fiems, and Jean-Marc Vincent, editors, *19th International Conference on Analytical and Stochastic Modelling Techniques and Applications (ASMTA'12)*, volume 7314 of *Lecture Notes in Computer Science (LNCS)*, pages 271–285, Grenoble, France, 4–6 July 2012. Springer.

[HRTW13] Gábor Horváth, Philipp Reinecke, Miklós Telek, and Katinka Wolter. Efficient Generation of PH-distributed Random Variates. (Extended version, submitted for publication), January 2013.

[HS08] Thomas Hacker and Preston Smith. Building a Network Simulation Model of the TeraGrid Network. In *TeraGrid 2008 Conference*, Las Vegas, NV, June 9-13 2008.

[HSXX] Øyvind Holmeide and Tor Skeie. Time Synchronization in Switched Ethernet. Technical report, OnTime Networks, XX. Unknown publication date. Available online: `http://www.westermo.com/dman/Document.phx/Manuals/Manuals+for+UK+only/Ontime/Articles/Time_Sync.pdf?folderId=%2FManuals%2FManuals+for+UK+only%2FOntime%2FArticles&cmd=download` (Last seen 14 February 2011).

[HT02] András Horváth and Miklós Telek. PhFit: A General Phase-Type Fitting Tool. In *TOOLS '02: Proceedings of the 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools*, pages 82–91, London, UK, 2002. Springer-Verlag.

[HT07] Gábor Horváth and Miklós Telek. A canonical representation of order 3 phase type distributions. In Katinka Wolter, editor, *Formal methods and stochastic models for Performance Evaluation*, volume 4748 of *LNCS*, pages 48–62, Berlin, Germany, 2007. Springer.

[HT08] Gábor Horváth and Miklós Telek. On the canonical representation of phase type distributions. *Performance Evaluation*, 2008.

[HT11] G. Horváth and M. Telek. Acceptance-rejection methods for generating random variates from matrix exponential distributions and rational arrival processes. In *Int. Conf. on Martix Analytic Methods (MAM)*, New York, New York, USA, june 2011.

[HZ06] Qi-Ming He and Hanqin Zhang. Spectral Polynomial Algorithms for Computing Bi-Diagonal Representations for Phase Type Distributions and Matrix-Exponential Distributions. *Stochastic Models*, 22:289–317, 2006.

[IEE08a] IEEE. Std 1588-2008, IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. `http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4579760`, 2008.

[IEE08b] The IEEE and The Open Group: The Open Group Base Specifications Issue 7. IEEE Std 1003.1-2008, 2008.

[IH05] Claudia Isensee and Graham Horton. Approximation of discrete phase-type distributions. In *Proceedings of the 38th annual Symposium on Simulation*, ANSS '05, pages 99–106, Washington, DC, USA, 2005. IEEE Computer Society.

[ITA95] The Internet Traffic Archive: NASA-HTTP – two months of HTTP logs from a busy WWW server. `http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html`, July 1995. (last seen 10 April 2012).

[ITU02] ITU. Recommendation Y.1540 – IP packet transfer and availability performance parameters. `http://www.itu.int/itudoc/itu-t/aap/sg13aap/history/y1540/y1540.html`, November 2002.

[ITU08] ITU. G.8261/Y.1361 (04/2008) Timing and synchronization aspects in packet networks . `http://www.itu.int/rec/T-REC-G.8261-200804-I`, April 2008.

[Jai91] Raj Jain. *The Art of Computer Systems Performance Analysis : Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley, New York :, 1991.

[JKB03] Kai S. Juse, Samuel Kounev, and Alejandro Buchmann. PetStore-WS: Measuring the Performance Implications of Web Services. In *Proceedings of the 29th International Conference of the Computer Measurement Group on Resource Management and Performance Evaluation of Enterprise Computing Systems (CMG 2003), Dallas, Texas, USA, December 7-12, 2003*, pages 113–123. Computer Measurement Group (CMG), December 2003.

[Job09] Sébastien Jobert. About the control of pdv using qos mechanisms and the applicability of proposed pdv metrics (mafe and mintdev). Technical report, France Telecom, 2009.

[Joh63] Selmer M. Johnson. Generation of Permutations by Adjacent Transposition. *Mathematics of Computation*, 17(83):282–285, July 1963.

[Jon09] Rick Jones. Netperf 2.4.5. `http://www.netperf.org/netperf/`, June 2009. (last seen 6 June 2011).

[JS03] F. Jarre and J. Stoer. *Optimierung*. Springer-Lehrbuch. Springer, 2003.

[JTD08] L. Juszczyk, Hong-Linh Truong, and S. Dustdar. Genesis - a framework for automatic generation and steering of testbeds of complex web servicesnet. In *Proc. 13th IEEE International Conference on Engineering of Complex Computer Systems ICECCS 2008*, pages 131–140, March 31 2008–April 3 2008.

[KB11] Jan Kriege and Peter Buchholz. Simulating Stochastic Processes with OMNeT++. In *Proceedings of the 4th International OMNeT++ Workshop (OMNeT++ 2011)*. ICST, 2011.

[KBF09] Ariane Keller, Rainer Baumann, and Ulrich Fiedler. TCN Trace Control for Netem. `http://tcn.hypert.net/`, 2009. (last seen 6 June 2011).

[KC07] Kenichi Kourai and Shigeru Chiba. A Fast Rejuvenation Technique for Server Consolidation with Virtual Machines. In *DSN '07: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 245–255, Washington, DC, USA, 2007. IEEE Computer Society.

[Klo11] Wouter Klouwen. Dhttpd/1.02a. `http://sourceforge.net/projects/dhttpd/`, 2011. (last seen 6 June 2011).

[Knu97] Donald E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, 1997.

[KR01] Balachander Krishnamurthy and Jennifer Rexford. *Web Protocols and Practice*. Addison Wesley, 2001.

[KR04] Su Myeon Kim and Marcel Catalin Rosu. A Survey of Public Web Services. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference (Alternate track papers & posters)*, pages 312–313, New York, NY, USA, 2004. ACM.

[Kra12] Tilman Krauß. Implementierung eines Werkzeugs zur Anpassung von Phasentyp-Verteilungen an Messwerte, 2012. Bachelor Thesis, Freie Universität Berlin (in German).

[KRB+12] Samuel Kounev, Philipp Reinecke, Fabian Brosig, Jeremy T. Bradley, Kaustubh Joshi, Vlastimil Babka, Anton Stefanek, and Stephen Gilmore. Providing Dependability and Resilience in the Cloud: Challenges and Opportunities. In Katinka Wolter, Alberto Avritzer, Marco Vieira, and Aad van Moorsel, editors, *Resilience Assessment and Evaluation of Computing Systems*, chapter 4, pages 65–81. Springer, 2012.

[LA96] A. Lang and J.L. Arthur. Parameter Approximation for Phase-Type Distributions. *Matrix-Analytic Methods in Stocastic Modells*, 183:151–206, 1996.

[LAJ99] Craig Labovitz, Abha Ahuja, and Farnam Jahanian. Experimental Study of Internet Stability and Backbone Failures. In *FTCS '99: Proceedings of the Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing*, page 278, Washington, DC, USA, 1999. IEEE Computer Society.

[Law06] A.M. Law. *Simulation Modeling and Analysis*. McGraw-Hill series in industrial engineering and management science. McGraw-Hill, 4th edition, 2006.

[Llo82] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–136, 1982.

[LLRG03] M.A. Luersen, R. Le Riche, and F. Guyon. A constrained, globalized, and bounded neldermead method for engineering optimization. *Structural and Multidisciplinary Optimization*, 27(1-2):43–54, May 2003.

[LMG95] D. Long, A. Muir, and R. Golding. A Longitudinal Survey of Internet Host Reliability. In *SRDS '95: Proceedings of the 14th Symposium on Reliable Distributed Systems*, page 2, Washington, DC, USA, 1995. IEEE Computer Society.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[LXM07] Nik Looker, Jie Xu, and Malcolm Munro. Determining the dependability of Service-Oriented Architectures. *International Journal of Simulation and Process Modelling*, 3(1/2):88–97, 2007.

[MC99] Stefanita Mocanu and Christian Commault. Sparse Representations of Phase-type Distributions. *Commun. Stat., Stochastic Models*, 15(4):759 – 778, 1999.

[ML03] Cleve Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45(1):3–49, 2003.

[Mon08] David Monniaux. The pitfalls of verifying floating-point computations. *ACM Trans. Program. Lang. Syst.*, 30(3):12:1–12:41, May 2008.

[MP02] Matthew Merzbacher and Dan Patterson. Measuring End-User Availability on the Web: Practical Experience. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 473–477, Washington, DC, USA, 2002. IEEE Computer Society.

[MR93] Manish Malhotra and Andrew Reibman. Selecting and implementing phase approximations for semi-markov models. *Communications in Statistics. Stochastic Models*, 9(4):473–506, 1993.

[MZ09a] Adele Marshall and M Zenga. Recent developments in fitting coxian phase-type distributions in healthcare. In *The XIII International Conference "Applied Stochastic Models and Data Analysis" (ASMDA-2009)*, pages 482–485, 2009.

[MZ09b] Adele H. Marshall and Mariangela Zenga. Simulating coxian phase-type distributions for patient survival. *International Transactions in Operational Research*, 16(2):213–226, 2009.

[NA04] R. Nossenson and H. Attiya. The distribution of file transmission duration in the web: Research articles. *Int. J. Commun. Syst.*, 17(5):407–419, 2004.

[Neu81] Marcel F. Neuts. *Matrix-Geometric Solutions in Stochastic Models. An Algorithmic Approach*. Dover Publications, Inc., New York, 1981.

[NM65] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, 1965.

[NP81] Marcel F. Neuts and Miriam E. Pagano. Generating random variates from a distribution of phase type. In *WSC '81: Proceedings of the 13th Winter Simulation Conference*, pages 381–387, Piscataway, NJ, USA, 1981. IEEE Press.

[ns2] The Network Simulator ns-2. http://www.isi.edu/nsnam/ns/. (last seen May 11, 2010).

[NS07] Nicholas Nethercote and Julian Seward. Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation. In *Proceedings of ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation (PLDI 2007)*, June 2007.

[O'C90] Colm Art O'Cinneide. Characterization of Phase-Type Distributions. *Stochastic Models*, 6:1–57, 1990.

[O'C91] Colm Art O'Cinneide. Phase-Type Distributions and Invariant Polytopes. *Advances in Applied Probability*, 23(3):515–535, 1991.

[OGP03] David Oppenheimer, Archana Ganapathi, and David A. Patterson. Why do Internet Services Fail, and What can be done about it? In *USITS'03: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*, pages 1–1, Berkeley, CA, USA, 2003. USENIX Association.

[OIY+09] Simon Ostermann, Alexandru Iosup, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing. In *1st International Conference on Cloud Computing*. ICST Press, 2009. Accepted for publication.

[PAB+05] Ruoming Pang, Mark Allman, Mike Bennett, Jason Lee, Vern Paxson, and Brian Tierney. A First Look at Modern Enterprise Traffic. In *IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 2–2, Berkeley, CA, USA, 2005. USENIX Association.

[PHA+04] Jeffrey Pang, James Hendricks, Aditya Akella, Roberto De Prisco, Bruce Maggs, and Srinivasan Seshan. Availability, Usage, and Deployment Characteristics of the Domain Name System. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 1–14, New York, NY, USA, 2004. ACM.

[PR06] Juan F. Pérez and Germán Riaño. jPhase: An Object-Oriented Tool for Modeling Phase-Type Distributions. In *Proceeding from the 2006 workshop on Tools for solving structured Markov chains*, SMCtools '06, New York, NY, USA, 2006. ACM.

[Pul09] Reza Pulungan. *Reduction of Acyclic Phase-Type Representations*. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 2009.

[PXL+04] Vasileios Pappas, Zhiguo Xu, Songwu Lu, Daniel Massey, Andreas Terzis, and Lixia Zhang. Impact of Configuration Errors on DNS Robustness. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 319–330, New York, NY, USA, 2004. ACM.

[R D06] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2006. ISBN 3-900051-07-0.

[RBD+09] Eric Rozier, Wendy Belluomini, Veera Deenadhayalan, Jim Hafner, K. K. Rao, and Pin Zhou. Evaluating the Impact of Undetected Disk Errors in RAID Systems. In *Proceedings of the 2009 IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2009, Estoril, Lisbon, Portugal, June 29 - July 2, 2009*, pages 83–92. IEEE, June/July 2009.

[RBD12] Philipp Reinecke, Levente Bodrog, and Alexandra Danilkina. Phase-type Distributions. In Katinka Wolter, Alberto Avritzer, Marco Vieira, and Aad van Moorsel, editors, *Resilience Assessment and Evaluation of Computing Systems*, chapter 5, pages 85–113. Springer, 2012.

[RDS02] Alma Riska, Vesselin Diev, and Evgenia Smirni. Efficient fitting of long-tailed data sets into phase-type distributions. *SIGMETRICS Perform. Eval. Rev.*, 30:6–8, December 2002.

[RDW11] Philipp Reinecke, Matthias Dräger, and Katinka Wolter. NetemCG – IP Packet-Loss Injection using a Continuous-Time Gilbert Model. Technical Report TR-B-11-05, Freie Universität Berlin, October 2011.

[RH12] Philipp Reinecke and Gábor Horváth. Phase-type Distributions for Realistic Modelling in Discrete-Event Simulation. In *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, SIMUTOOLS'12, pages 283–290, ICST, Brussels, Belgium, 2012. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[RKW12a] Philipp Reinecke, Tilman Krauß, and Katinka Wolter. Cluster-based fitting of phase-type distributions to empirical data. *Computers & Mathematics with Applications*, 64(12):3840–3851, December 2012. Special Issue on Theory and Practice of Stochastic Modeling.

[RKW12b] Philipp Reinecke, Tilman Krauß, and Katinka Wolter. HyperStar: Phase-Type Fitting Made Easy. In *9th International Conference on the Quantitative Evaluation of Systems (QEST) 2012*, pages 201–202, September 2012. Tool Presentation.

[RLT78] B. Randell, P. Lee, and P. C. Treleaven. Reliability Issues in Computing System Design. *ACM Computing Surveys*, 10(2):123–165, 1978.

[RTW10] Philipp Reinecke, Miklós Telek, and Katinka Wolter. Reducing the Costs of Generating APH-Distributed Random Numbers. In B. Müller-Clostermann, K. Echtle, and E. Rathgeb, editors, *MMB & DFT 2010*, number 5987 in LNCS, pages 274–286. Springer-Verlag Berlin Heidelberg, 2010.

[RvMW04] Philipp Reinecke, Aad P. A. van Moorsel, and Katinka Wolter. A Measurement Study of the Interplay Between Application Level Restart and Transport Control Protocol. In Miroslaw Malek, Manfred Reitenspieß, and Jörg Kaiser, editors, *Service Availability. Proceedings of the First International Service Availability Symposium*, volume 3335 of *LNCS*, pages 86–100. Springer, May 2004.

[RvMW06a] Philipp Reinecke, Aad P. A. van Moorsel, and Katinka Wolter. Experimental Analysis of the Correlation of HTTP GET Invocations. In András Horváth and Miklós Telek, editors, *Formal Methods and Stochastic Models for Performance Evaluation*, volume 4054 of *LNCS*, pages 226–237. Springer, June 2006.

[RvMW06b] Philipp Reinecke, Aad P. A. van Moorsel, and Katinka Wolter. The Fast and the Fair: A Fault-Injection-Driven Comparison of Restart Oracles for Reliable Web Services. In *QEST '06: Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems*, pages 375–384, Washington, DC, USA, 2006. IEEE Computer Society.

[RW08a] Philipp Reinecke and Katinka Wolter. Adaptivity Metric and Performance for Restart Strategies in Web Services Reliable Messaging. In *WOSP '08: Proceedings of the 7th International Workshop on Software and Performance*, pages 201–212, New York, NY, USA, 2008. ACM.

[RW08b] Philipp Reinecke and Katinka Wolter. Phase-Type Approximations for Message Transmission Times in Web Services Reliable Messaging. In Samuel Kounev, Ian Gorton, and Kai Sachs, editors, *Performance Evaluation – Metrics, Models and Benchmarks*, volume 5119 of *Lecture Notes in Computer Science*, pages 191–207. Springer, June 2008.

[RW08c] Philipp Reinecke and Katinka Wolter. Towards a multi-level fault-injection test-bed for service-oriented architectures: Requirements for parameterisation. In *SRDS Workshop on Sharing Field Data and Experiment Measurements on Resilience of Distributed Computing Systems*, Naples, Italy, 2008. AMBER.

[RW10] Philipp Reinecke and Katinka Wolter. A Simulation Study on the Effectiveness of Restart and Rejuvenation to Mitigate the Effects of Software Ageing. In *WoSAR 2010*, 2010.

[RW11] Philipp Reinecke and Katinka Wolter. On Stochastic Fault-Injection for IP-Packet Loss Emulation. In Nigel Thomas, editor, *Computer Performance Engineering. Proceedings of the 8th European Performance*

Engineering Workshop, EPEW 2011, number 6977 in LNCS. Springer, October 2011.

[RWBT09] Philipp Reinecke, Katinka Wolter, Levente Bodrog, and Miklos Telek. On the Cost of Generating PH-distributed Random Numbers. In Gábor Horváth, Kaustubh Joshi, and Armin Heindl, editors, *Proceedings of the Ninth International Workshop on Performability Modeling of Computer and Communication Systems (PMCCS-9)*, pages 16–20, Eger, Hungary, September 17–18, 2009 2009.

[RWM10] Philipp Reinecke, Katinka Wolter, and Miroslaw Malek. A Survey on Fault-Models for QoS Studies of Service-Oriented Systems. Technical Report B-2010-02, Freie Universität Berlin, February 2010.

[RWW09] Philipp Reinecke, Sebastian Wittkowski, and Katinka Wolter. Response-time Measurements Using the Sun Java Adventure Builder. In *QUASOSS '09: Proceedings of the 1st International Workshop on Quality of Service-oriented Software Systems*, pages 11–18, New York, NY, USA, 2009. ACM.

[SF07] Nicolas Salatge and Jean-Charles Fabre. Fault tolerance connectors for unreliable web services. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN '07, pages 51–60, Washington, DC, USA, 2007. IEEE Computer Society.

[SG06] Bianca Schroeder and Garth A. Gibson. A Large-scale Study of Failures in High-performance Computing Systems. In *DSN '06: Proceedings of the International Conference on Dependable Systems and Networks*, pages 249–258, Washington, DC, USA, 2006. IEEE Computer Society.

[SG07] Bianca Schroeder and Garth A. Gibson. Disk failures in the real world: what does an MTTF of 1,000,000 hours mean to you? In *FAST '07: Proceedings of the 5th USENIX conference on File and Storage Technologies*, page 1, Berkeley, CA, USA, 2007. USENIX Association.

[SH08] R. Sadre and B.R.H.M. Haverkort. Fitting heavy-tailed http traces with the new stratified em-algorithm. In *4th International Telecommunication Networking Workshop on QoS in Multiservice IP Networks (IT-NEWS)*, pages 254–261, Los Alamitos, February 2008. IEEE Computer Society Press.

[SLO10] S. Salsano, F. Ludovici, and A. Ordine. Definition of a general and intuitive loss model for packet networks and its implementation in the Netem module in the Linux kernel. Technical report, University of Rome "Tor Vergata", September 2010.

[SMS06] L. Silva, H. Madeira, and J.G. Silva. Software Aging and Rejuvenation in a SOAP-based Server. In *Proc. Fifth IEEE International Symposium on Network Computing and Applications NCA 2006*, pages 56–65, 2006.

[SN09] S. Singer and J. Nelder. Nelder-mead algorithm. *Scholarpedia*, 4(2):2928, 2009.

[Ste69] Laurence Sterne. *The Life and Opinions of Tristram Shandy, Gentleman.* Wordsworth Editions Limited, 1996 edition, 1757–1769.

[Ste09] William J. Stewart. *Probability, Markov Chains, Queues and Simulation. The Mathematical Basis of Performance Modeling.* Princeton University Press, 2009.

[Sun06] Sun Microsystems. SUN Java Adventure Builder Reference Application. `https://adventurebuilder.dev.java.net/`, 2006. Last seen April 28th, 2009.

[TBT06] Axel Thümmler, Peter Buchholz, and Miklos Telek. A Novel Approach for Phase-Type Fitting with the EM Algorithm. *IEEE Trans. Dependable Secur. Comput.*, 3(3):245–258, 2006.

[TH02a] M. Telek and A. Heindl. Matching Moments for Acyclic Discrete and Continous Phase-Type Distributions of Second Order. *International Journal of Simulation Systems, Science & Technology*, 3(3–4):47–57, December 2002.

[TH02b] M. Telek and A. Heindl. Moment bounds for acyclic discrete and continuous phase-type distributions of second order. In *Proc. of UK Performance Evaluation Workshop*, 2002.

[The08] The Apache Software Foundation. Apache JMeter. `http://jakarta.apache.org/jmeter`, 2008. (last seen 11 April 2012).

[TKDT04] Dong Tang, Dileep Kumar, Sreeram Duvur, and Oystein Torbjornsen. Availability Measurement and Modeling for An Application Server. In *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks*, page 669, Washington, DC, USA, 2004. IEEE Computer Society.

[TVN+03] M. Tian, T. Voigt, T. Naumowicz, H. Ritter, and J. Schiller. Performance Impact of Web Services on Internet Servers. In *Proceedings of IASTED International Conference on Parallel and Distributed Computing and Systems*, Marina Del Rey, California, USA, 2003.

[Var01] András Varga. The OMNeT++ Discrete Event Simulation System. In *Proceedings of the European Simulation Multiconference (ESM'2001)*, June 2001.

[VM02] Marco Vieira and Henrique Madeira. Definition of Faultloads Based on Operator Faults for DMBS Recovery Benchmarking. In *PRDC '02: Proceedings of the 2002 Pacific Rim International Symposium on Dependable Computing*, page 265, Washington, DC, USA, 2002. IEEE Computer Society.

[VM03] Marco Vieira and Henrique Madeira. A Dependability Benchmark for OLTP Application Environments. In *VLDB '2003: Proceedings of the 29th international conference on Very large data bases*, pages 742–753. VLDB Endowment, 2003.

[vMAB+09] Aad van Moorsel, Eugenio Alberdi, Raul Barbosa, Robin Bloomfield, Andrea Bondavalli, João Durães, Rosaria Esposito, Lorenzo Falai, Johan Karlsson, Paolo Lollini, Henrique Madeira, Istvan Majzik, Zoltán Micskei, Leonardo Montecchi, Gergely Pinter, Vladimir Stankovic, Lorenzo Strigini, Michele Vadursi, Marco Vieira, Katinka Wolter, and Huqiu Zhang. State of the art. Technical Report D2.2, Assessing, Measuring, and Benchmarking Resilience (AMBER), 30 June 2009.

[vMBP+08] Aad van Moorsel, Andrea Bondavalli, Gergely Pinter, Henrique Madeira, Istvan Majzik, João Durães, Johan Karlsson, Lorenzo Falai, Lorenzo Strigini, Marco Vieira, Michele Vadursi, Paolo Lollini, and Rosaria Esposito. State of the art. Technical Report D2.1, Assessing, Measuring, and Benchmarking Resilience (AMBER), April 2008.

[vMW06] A. P. A. van Moorsel and K. Wolter. Analysis of Restart Mechanisms in Software Systems. *IEEE Transactions on Software Engineering*, 32(8), August 2006.

[WBPG09] Guanying Wang, Ali Raza Butt, Prashant Pandey, and Karan Gupta. A Simulation Approach to Evaluating Design Decisions in MapReduce Setups. In *2009 IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 1–11. IEEE, 2009.

[WLS08] Junfeng Wang, Jin Liu, and Chundong She. Segment-based adaptive hyper-erlang model forlong-tailed network traffic approximation. *The Journal of Supercomputing*, 45:296–312, 2008. 10.1007/s11227-008-0173-5.

[Wol10] Katinka Wolter. *Stochastic Models for Fault Tolerance - Restart, Rejuvenation and Checkpointing*. Springer Verlag, 2010.

[WR08] Katinka Wolter and Philipp Reinecke. Restart in competitive environments. Technical Report DTR08-9, Department of Computing, Imperial College London, Department of Computing, Imperial College London, Huxley Building, 180 Queen's Gate, London SW7 2RH, July 2008.

[WR10] Katinka Wolter and Philipp Reinecke. Stochastic models for dependable services. *Electronic Notes in Theoretical Computer Science*, 261:5–21, 2010.

[WRM11] Katinka Wolter, Philipp Reinecke, and Alfons Mittermaier. Model-based Evaluation and Improvement of PTP Syntonisation Accuracy in Packet-Switched Backhaul Networks for Mobile Applications. In Nigel Thomas, editor, *Computer Performance Engineering. Proceedings of the 8th European Performance Engineering Workshop, EPEW 2011*, number 6977 in LNCS, pages 219–234. Springer, October 2011.

[WZXL05] Junfeng Wang, Hongxia Zhou, Fanjiang Xu, and Lei Li. Hyper-erlang based model for network traffic approximation. In Yi Pan, Daoxu Chen, Minyi Guo, Jiannong Cao, and Jack Dongarra, editors, *Parallel and Distributed Processing and Applications*, volume 3758 of *Lecture Notes in Computer Science*, pages 1012–1023. Springer Berlin / Heidelberg, 2005. 10.1007/11576235_101.

[XP09] Kaiqi Xiong and Harry Perros. Service Performance and Analysis in Cloud Computing. In *SERVICES '09: Proceedings of the 2009 Congress on Services - I*, pages 693–700, Washington, DC, USA, 2009. IEEE Computer Society.

[YMKT99] M. Yajnik, Sue Moon, J. Kurose, and D. Towsley. Measurement and Modelling of the Temporal Dependence in Packet Loss. In *Proc. IEEE Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies INFOCOM '99*, volume 1, pages 345–352, 1999.

[ZDPS01] Yin Zhang, Nick Duffield, Vern Paxson, and Scott Shenker. On the Constancy of Internet Path Properties. In *IMW '01: Proceedings of*

the 1st ACM SIGCOMM Workshop on Internet Measurement, pages 197–211, New York, NY, USA, 2001. ACM.

[ZF07] Ming Zhao and Renato J. Figueiredo. Experimental Study of Virtual Machine Migration in Support of Reservation of Cluster Resources. In *VTDC '07: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, pages 1–8, 2007.

[ZHB10] Ryan Zarick, Mikkel Hagen, and Radim Bartos. The impact of network latency on the synchronization of real-world ieee 1588-2008 devices. In *Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2010 International IEEE Symposium on*, pages 135 –140, October 2010.

[ZL08] Zibin Zheng and Michael R. Lyu. A distributed replication strategy evaluation and selection framework for fault tolerant web services. In *ICWS*, pages 145–152. IEEE Computer Society, 2008.

[ZPS00] Y. Zhang, V. Paxson, and S. Shenker. The Stationarity of Internet Path Properties: Routing, Loss, and Throughput. *ACIRI Technical Report*, 2000.

[ZZL10] Zibin Zheng, Yilei Zhang, and Michael R. Lyu. Distributed qos evaluation for real-world web services. In *Proceedings of the 2010 IEEE International Conference on Web Services*, ICWS '10, pages 83–90, Washington, DC, USA, 2010. IEEE Computer Society.