COMPUTER AND COMMUNICATION SCIENCES

# PERFORMANCE EVALUATION OF COMPUTER AND COMMUNICATION SYSTEMS

Jean-Yves Le Boudec

EPFL Press
Distributed by CRC Press

COMPUTER AND COMMUNICATION SCIENCES

# PERFORMANCE EVALUATION OF COMPUTER AND COMMUNICATION SYSTEMS

Jean-Yves Le Boudec



EPFL Press
A Swiss academic publisher distributed by CRC Press

**Taylor and Francis Group, LLC**
6000 Broken Sound Parkway, NW, Suite 300,
Boca Raton, FL 33487

**Distribution and Customer Service**
orders@crcpress.com

**www.crcpress.com**

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the Library of Congress.

This book is published under the editorial direction of
Professor Serge Vaudenay (EPFL).

The authors and publisher express their thanks to the Ecole polytechnique fédérale de Lausanne (EPFL) for its generous support towards the publication of this book.

Cover artwork: Elias Le Boudec

EPFL Press

is an imprint owned by Presses polytechniques et universitaires romandes, a Swiss academic publishing company whose main purpose is to publish the teaching and research works of the Ecole polytechnique fédérale de Lausanne (EPFL) and other universities and institutions of higher learning.

Presses polytechniques et universitaires romandes
EPFL – Rolex Learning Center
Post office box 119
CH-1015 Lausanne, Switzerland
E-mail: ppur@epfl.ch
Phone: 021/693 21 30
Fax: 021/693 40 27

**www.epflpress.org**

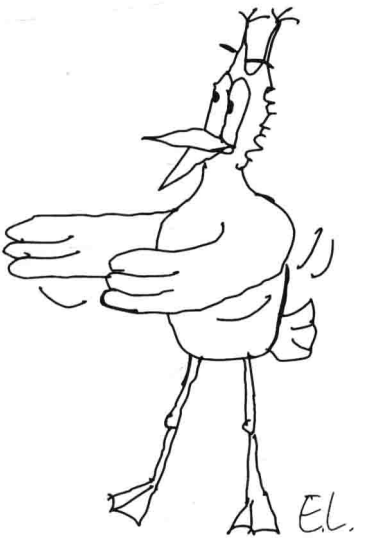

ISBN 978-2-940222-40-7 (EPFL Press)
ISBN 978-1-4398-4992-7 (CRC Press)

Printed in Italy

# PERFORMANCE EVALUATION OF COMPUTER AND COMMUNICATION SYSTEMS

# Preface

Performance evaluation is often the critical part of evaluating the results of a research project. Many of us are familiar with simulations, but it is frequently difficult to address questions like: Should I eliminate the beginning of the simulation in order for the system to become stabilized? I simulate a random way-point model but the average speed in my simulation is not as expected. What happened? The reviewers of my study complained that I did not provide confidence intervals. How do I go about this? I would like to characterize the fairness of my protocol. Should I use Jain's Fairness Index or the Lorenz Curve Gap? I would like to fit a distribution to the flow sizes that I measured, but all my measurements are truncated to a maximum value; how do I account for the truncation?




This book groups a set of lecture notes for a course given at EPFL. It contains all the material needed by an engineer who wishes to evaluate the performance of a computer or communication system. More precisely, with this book and some accompanying practicals, you will be able to answer the above and other questions, evaluate the performance of computer and communication systems and master the theoretical foundations of performance evaluation and of the corresponding software packages.

In the past, many textbooks on performance evaluation have given the impression that this is a complex field, with large amounts of baroque queuing theory excursions, which can be exercised only by performance evaluation experts. This is not necessarily the case. In contrast, performance evaluation can and should be performed by any computer engineering specialist who designs a system. When a plumber installs pipes in our house, one expects her to properly size their diameters; the same holds for computer engineers.

This book is not intended for the performance evaluation specialist. It is adressed *to any computer engineer or scientist* who is active in the development or operation of software or hardware systems. The required background is an elementary course in probability and one in calculus.

The objective of this book is therefore to make performance evaluation usable by all computer engineers and scientists. The foundations of performance

evaluation reside in statistics and queuing theory. Therefore, *some* mathematics are involved and the text cannot be overly simplified. However, it turns out that much of the complications are not in the general theories, but in the exact solution of specific models. For example, certain textbooks on statistics (but none of the ones cited in the reference list) develop various solution techniques for specific models, the vast majority of which are encapsulated in commercially or freely available software packages like Matlab, S-PLUS, Excel, Scilab or R.

To avoid this pitfall, we focus first on the *what* before the *how*. Indeed, the most difficult question in a performance analysis is often "what to do"; once you know what to do, it is less difficult to find a way with your usual software tools or by shopping the web. For example, what do we do when we fit a model to data using least square fitting (Chapter 3)? What is a confidence interval? What is a prediction interval (Chapter 2)? What is the congestion collapse pattern (Chapter 1)? What is the null hypothesis in a test and what does the result of a test *really* mean (Chapter 4)? What is an information criterion (Chapter 5)? If no failure appears out of $n$ experiments, what confidence interval can I give for the failure probability (Chapter 2)?

Second, regarding the *how*, we looked for solution methods that are as universal as possible, i.e. that apply to many situations, whether simple or complex. There are several reasons for this. Firstly, one should use only methods and tools that one understands, and a good engineer should first invest some time in learning tools and methods that he/she will use more often. Secondly, brute force and a computer can do a lot more than one often seems to believe. This philosophy is in sharp contrast to some publications on performance evaluation. For example, computing confidence or prediction intervals can be made simple and systematic if we use the median and not the mean; if we have to employ the mean, the use of the likelihood ratio statistic is quite universal and requires little intellectual sophistication regarding the model. Thus, we focus on generic methods such as: the use of filters for forecasting (Chapter 5), bootstrap and Monte-Carlo simulations for evaluating averages or prediction intervals (Chapter 6), the likelihood ratio statistic for tests (Chapter 2, Chapter 4), importance sampling (Chapter 6), least-square and $\ell^1$-norm minimization methods (Chapter 3).

When presenting solutions, we try *not* to hide their limitations and the cases where they do not work. Indeed, some frustrations experienced by young researchers can sometimes be attributed to false expectations about the power of various methods.

We give a coverage of queuing theory that attempts to strike a balance between depth and relevance. During a performance analysis, one is often confronted with the dilemma: should we use an approximate model for which exact solutions exist, or approximate solutions for a more exact model? We propose four topics (deterministic analysis, operational laws, single queues, queuing networks) which provide a good balance. We illustrate in a case study how the four topics can be utilized to provide different insights on a queuing question. For queuing networks, we give a unified treatment, which is perhaps the first of its kind at this level of synthesis. We show that complex topics such as

queues with concurrency (MSCCC queues) or networks with bandwidth sharing (Whittle networks) all fit in the same framework of product form queuing networks. Results of this kind have been traditionally presented as separate; unifying them simplifies the student's job and provides new insights.

We develop the topic of Palm calculus, also called "the importance of the viewpoint", which is so central to queuing theory, as a topic of its own. Indeed, this topic has so many applications to simulation and to system analysis in general, that it is a very good time investment. Here too, we focus on general purpose methods and results, in particular the large-time heuristic for mapping various viewpoints (Chapter 7).

Chapter 1 gives a methodology and serves as an introduction to the rest of the book. Performance patterns are also described, i.e. facts that repeatedly appear in various situations, and the knowledge of which considerably helps the performance evaluation.

Chapter 2 demonstrates how to summarize experimental or simulation results, as well as how to quantify their accuracy. It also serves as an introduction to a scientific use of the statistical method, i.e. pose a model and verify its assumptions. In Chapter 3, we present general methods for fitting an explanatory model to data and the concept of heavy tail. Chapter 4 describes the techniques of tests, and Chapter 5 those of forecasting. These four chapters give a coverage of modern statistics useful to our field.

Chapter 6 discusses discrete event simulation and several important, though simple issues such as the need for transient removal, for confidence intervals, and classical simulation techniques. We also discuss importance sampling, which is very useful for computing estimates of rare events; we give a simple, though quite general and broadly applicable method.

Chapter 7 describes Palm calculus, which relates the varying viewpoints resulting from measurements done by different operators. Here, we discuss freezing simulations, a phenomenon which can be a problem for even simple simulations if one is not aware of it. We also present how to perform a perfect simulation of stochastic recurrences. Chapter 8 discusses patterns specific to queuing, classical solution methods for queuing networks, and, perhaps more important, operational analysis for rapid evaluation.

The appendix gives background information that cannot yet be easily found elsewhere, such as a Fourier-free quick crash course on digital filters (used in Chapter 5) and confidence intervals for quantiles.

Performance evaluation is primarily an art, and involves using sophisticated tools such as mathematical packages, measurement tools and simulation tools. See the web site of the EPFL lecture on Performance Evaluation for some examples of *practicals*, implemented in Matlab and designed around this book.

The text is intended for self-study. Proofs are not given when there are easily accessible references (these are indicated in the text); otherwise they can be found in appendixes at the end of the chapters.

The *Index* collects all terms and expressions that are highlighted in the text like ***this*** and also serves as a notation list.

**Acknowledgements**

I would like to thank Anthony Davison for allowing me to access a beta version of his book "Statistical Models" as well as Richard Weber, who made his lecture notes freely available on the web and allowed me to use them as constituent material in an early version of this course. I am grateful to François Baccelli and Pierre Brémaud who helped me obtain some understanding of their fields. Many thanks go to Mourad Kara for discussions and input, to Irina Baltcheva, Manuel Flury, Olivier Gallay, Assane Gueye, Paul Hurley, Ruben Merz, Božidar Radunović, Gianluca Rizzo, Slaviša Sarafijanović, Milan Vojnović, Utkarsh Upadhyay and Jonas Wagner for various input and comments. I thank Scouac, Pilou and their friends for visiting our pages here and there. Last but not least, I thank Elias for the artwork.

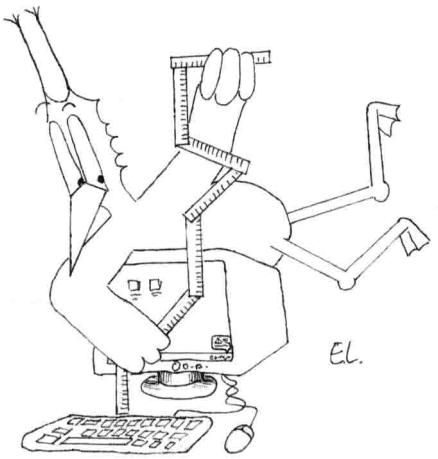
Jean-Yves Le Boudec, EPFL

# Contents

CHAPTER 1

# Methodology

Perhaps the most difficult part when carrying out a performance evaluation is knowing where to start. In this chapter, we propose a methodology, i.e. a set of recommendations, that is valid for any performance evaluation study. We stress the importance of factors, in particular hidden ones, and the need to use the scientific method. We also discuss a few frequent performance patterns, as a means of quickly focusing on important issues.




## 1.1 What is Performance Evaluation?

In the context of this book, performance evaluation involves quantifying the service delivered by a computer or a communication system. For example, we might be interested in: comparing the power consumption of several server farm configurations; knowing the response time experienced by a customer performing a reservation over the Internet; comparing compilers for a multiprocessor machine.

In all cases it is important to carefully define the *load* and the *metric*, and to be aware of the performance evaluation *goals*.

### 1.1.1 Load

An important feature of computer or communication systems is that their performance depends dramatically on the ***workload*** (or simply ***load***) that they are subjected to. The load characterizes the quantity and the nature of requests submitted to the system. Consider for instance the problem of quantifying the performance of a web server. We could characterize the load by a simple concept such as the number of requests per second. This is called the ***intensity of the workload***. In general, the performance deteriorates when the intensity increases, but this deterioration is often sudden. The reason for this is the non-linearity of queuing systems – a ***performance pattern*** example that is discussed in Section 1.5 and Chapter 8.

The performance of a system depends not only on the intensity of the workload, but also on its nature. On a web server, for example, all requests are not equivalent: some web server softwares might perform well with *get* requests for frequently used objects, and less well with requests that require database access. For other web servers, things might be different. This issue is addressed by using standardized mixes of web server requests. These are generated by a ***benchmark***, defined as a load generation process that intends to mimic a typical user behavior. Chapter 3 presents a study of how such a benchmark can be constructed.

### 1.1.2 Metric

A performance ***metric*** is a measurable quantity that precisely captures what we want to measure – it can take on many forms. There is no general definition of a performance metric: it is system dependent, and its definition requires a good understanding of the system and its users. We will often mention examples where the metric corresponds to throughput (the number of tasks completed per unit of time), power consumption (the integral of the electrical energy consumed by the system, per time unit), or response time (i.e. the time elapsed between a start and an end event). For each performance metric, we may be interested in the average, 95-percentile, worst-case, etc., as explained in Chapter 2.

**Example 1.1 Windows versus Linux**

Chen and co-authors compare Windows to Linux in [25]. As metric, they use the number of CPU cycles, the number of instructions, the number of data read/write operations required by a typical job. The load was generated by various benchmarks: "syscall" generates elementary operations (system calls); "memory read" generates references to an array; an application benchmark runs a popular application.

It is also important to be aware of the experimental conditions under which the metric is measured, as illustrated by the following example.