Performance Evaluation by Simulation and Analysis
with Applications to Computer Networks

Series Editor
Jean-Charles Pomerol

Performance Evaluation by Simulation and Analysis with Applications to Computer Networks

Ken Chen

ISTE

WILEY

First published 2015 in Great Britain and the United States by ISTE Ltd and John Wiley & Sons, Inc.

Apart from any fair dealing for the purposes of research or private study, or criticism or review, as permitted under the Copyright, Designs and Patents Act 1988, this publication may only be reproduced, stored or transmitted, in any form or by any means, with the prior permission in writing of the publishers, or in the case of reprographic reproduction in accordance with the terms and licenses issued by the CLA. Enquiries concerning reproduction outside these terms should be sent to the publishers at the undermentioned address:

ISTE Ltd
27-37 St George's Road
London SW19 4EU
UK

www.iste.co.uk

John Wiley & Sons, Inc.
111 River Street
Hoboken, NJ 07030
USA

www.wiley.com

© ISTE Ltd 2015
The rights of Ken Chen to be identified as the author of this work have been asserted by him in accordance with the Copyright, Designs and Patents Act 1988.

Library of Congress Control Number: 2014958253

British Library Cataloguing-in-Publication Data
A CIP record for this book is available from the British Library
ISBN 978-1-84821-747-8

Contents

List of Tables

List of Figures

List of Listings

Preface

When, as a PhD candidate, I discovered the technique of computer-based simulation through the book of G. Fishman [FIS 73] in the documentation centre of INRIA[1] (Rocquencourt, France), I was impressed by this numerical tool and its use for performance evaluation, because of its high degree of modeling capability, and also by it being relatively easy to learn with my knowledge in computer programming. By practising it, I knew very quickly that, behind its programming interface, this experimental tool is actually governed by a rigorous scientific background of statistics and the theory of probability. With years of practice, I understood that it is also an art.

This book primarily comes from lectures I give to students in the engineering faculty on Telecommunications and Computer networks, as well as those following the Master's degree curriculum on the same subject. The matter addressed in this book can also be useful for PhD students as well as engineers. This book chooses the computer networks as the target application domain. However, the tools presented here are generic, they are applicable to the computer systems in general, and more widely to many other application domains.

Computer networks are taking an increasingly significant place in modern society. More particularly, this book is focused on a crucial problem which is the mastering of the performance of computer networks. The latter are becoming increasingly complex systems, and their presence and importance

1 *Institut national de recherche en informatique et en automatique*, the French National Institute for Research in Computer and Control Sciences.

do not cease growing in all sectors. Thus, the control of these systems becomes a major issue and a common concern for many actors: network operators, Internet Service Providers (ISP), equipment manufacturers, etc.

We are interested here, more particularly, in the quantitative aspects of these systems, rather than their functional aspects. As an example, let us imagine that you are in charge of setting-up a Website to promote a product. You must of course have a good knowledge of adequate technologies (HTML, PHP, MySQL, etc.) to ensure the building of this Website. You must also choose the adequate hardware and configuration on which the Website will run. The quantitative aspect then plays a crucial role. Actually, the hardware and configuration must meet the expectation of potential customers. There is a real *trade-off* between:

– *performance to reach*: i.e. identification of some indicators (e.g. response time of the site), as well as the associated threshold values, related to the proper functioning of the Website and the user's satisfaction;

– *resource dimensioning*: i.e. the assessment of the physical resources (e.g. type of the server (processing power), amount of bandwidth) to be provisioned in order to reach these performances.

An over-sizing induces a useless additional cost, whereas a lack of provisioning could make potential customers unsatisfied. In both cases, there is a negative impact on competitiveness. It is thus necessary to establish a good adequacy between the required performance and the resources to be committed. This trade-off is not easy to reach and it claims a mastering of the quantitative aspects of the system. This is the objective of this book.

This book presents a set of theoretical results and practical methods which constitutes a panoply of complementary tools for the quantitative study of computer networks. The complementarity of these tools is an essential element for the author, since the systems to be studied are complex. This book was designed in this spirit and this is why it includes two facets:

– computer-based simulation with its languages and its practices;

– mathematical modeling with the queueing theory, and, more generally, the Markov processes.

This book wants to be pragmatic and devotes a significant part to the simulation since the latter presents a double advantage:

– It allows a detailed and faithful modeling;

– It is accessible with a minimum knowledge in computer programming without claiming advanced expertise in mathematics.

The author presents the practice of simulation through the `OMNeT++` simulation environment. This tool is chosen not only because of its rich functional features and its popularity, but also because of its accessibility: this *open source* software is available from the Internet and can be used freely (under some conditions similar to GNU GPL).

A study by simulation, as detailed as it could be, is basically a statistical study, for a particular configuration of the target system. However, the mathematical (or analytical) modeling offers firmly built results with theoretical background. These theoretical results are general and parametrizable, ready for study of various scenarios. However, the constraints of mathematical tractability often lead to a model which is less faithful than that of a simulator. So analytical modeling and simulation form a couple of complementary tools for the quantitative study of the systems.

The author also has a concern of making the theoretical bases of the performance evaluation available to a public with average mathematical level. For this reason, we choose to emphasize on concepts, rather than to focus on mathematical details. For this reason also, we include a recall of the principal concepts and results resulting from the theory of probability and statistics which are used, implicitly or explicitly, in this book.

The structure (see Figure P.1) of the book is presented below:

– we start with a short introduction to the problem of performance evaluation in computer networks. Then, we introduce the general principles of the performance evaluation and the principal tools, which are analytical modeling and computer-based simulation. We divide the continuation of the book into three parts, which are devoted respectively to simulation, analytical models and the underlying theoretical background (probability, statistics and Markov process);

– the first part (Chapters 2 to 6, page 13 to page 127) is focused on simulation. We deal, of course, with the principle and practice of the coding of a simulation program. We also emphasize fact that simulations are basically statistical experiments. So the design and the control of a simulation, as well

as the collection and the interpretation of the data, must follow precise rules which we will present. This part will be illustrated through examples carried out in OMNeT++;

– the second part (Chapters 7 to 11, page 131 to page 199) is devoted to modeling by queueing theory, and, more generally, stochastic processes. We present some basic theoretical results, including $M/M/.$ queues, $M/G/1$ and its variants (server vacation and priority queueing), as well as the queueing networks (mainly Jackson networks);

– in the third part (Chapters 12 to 14, page 203 to page 272), a short introduction to the theory of probability and statistics is provided. Indeed, both analytical modeling and simulation actually proceed to a modeling of the system as a stochastic system. Thus the statistics and the theory of probability constitute the theoretical background of these two approaches. This part also contains an introduction to the Markov processes. This part was conceived to minimize the repetitions of the same concept used in various sections of the book; it also aims to facilitate possible needs for refreshing certain mathematical concepts, it makes this manuscript autonomous (self-containing).

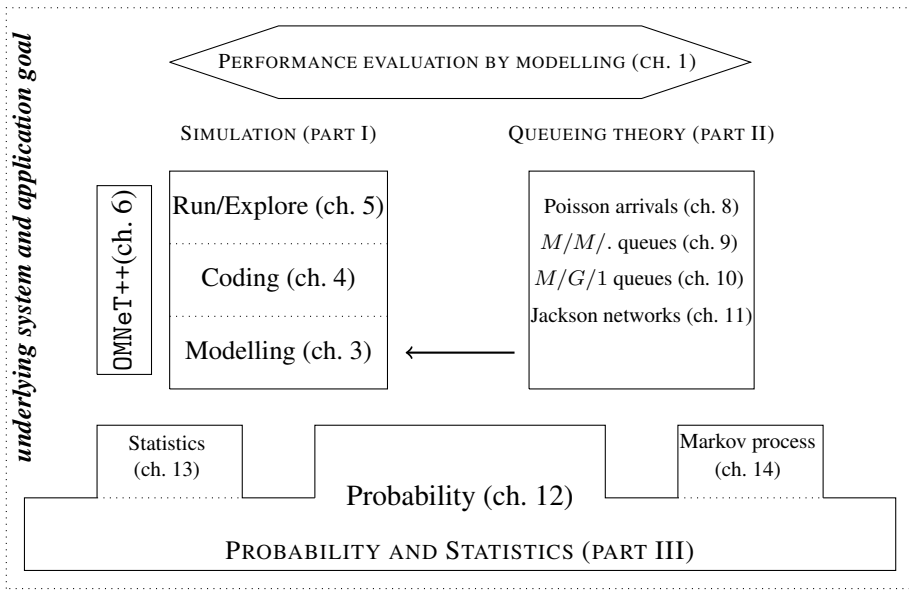

Figure P.1. *Outline of this book*

Over a total of about 272 pages of text, the simulation, i.e. Part I and Chapter 13 (Statistics), constitutes roughly 50%, the analytical modeling, i.e. Part II and Chapter 14 (Markov process), constitutes 35% and the basis of Probability (Chapter 12) 10%, the rest is occupied by introductory matters (see Figure P.2).

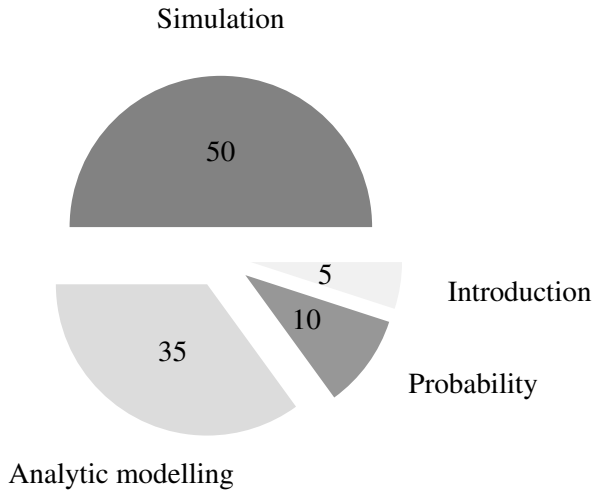

Figure P.2. *Different parts of this book*

In this book, an effort is made to facilitate the cross referencing (*à la* Html documents). Indeed, a printed book is by definition, a serialized document, whereas the matters addressed in this book contain many mutual overlaps.

Ken CHEN
December 2014

1

Performance Evaluation

In this chapter we introduce the performance evaluation jointly with the resource provisioning. We then give a survey on the general aspects of modeling before focusing on simulation and analytical modeling.

1.1. Performance evaluation

Recent developments in telecommunications are characterized by a double diversity: diversity of applications and diversity of infrastructures. New applications and services appear day by day. The amount of data to be transported is constantly increasing, of which multimedia data are taking an increasingly important part. These data are characterized by their tight time constraints, high flow rates and abrupt variations in time. We are witnessing the advent of the so-called *ambient intelligence*, i.e. computers and data are everywhere, they can be produced everywhere (cloud, body-area sensor, your smart phone, etc.) and consumed everywhere else (your laptop, your smart phone, a remote monitoring facility, etc.). This ubiquitous/pervasive computing environment is supported by various networking infrastructures, including Digital Subscriber Line (DSL) connections, 3G/4G cellular networks, wireless Local Area Network (LAN), optical networks, etc.

One important issue is the so-called *Quality of Service* (QoS) of the data transmission. Roughly speaking, the QoS required by an application is the set of parameters (throughput, delay, jitter, reliability, etc.) on which the application relies. For example, for a videoconferencing, the nominal end-to-end delay is about 150 ms. If the actual delay is too variable, such that

Performance Evaluation by Simulation and Analysis with Applications to Computer Networks, First Edition. Ken Chen. © ISTE Ltd 2015. Published by ISTE Ltd and John Wiley & Sons, Inc.

the delay is frequently higher than 150 ms, we simply cannot perform decent videoconferencing.

It is a fundamental matter to guarantee QoS for different applications. One of the key issues is the performance evaluation, and, in a dual manner, the resource provisioning. Performance evaluation aims to quantitatively assess various parameters of a system running most often in a stochastic environment.

A computer network is a complex system which is constituted of various components organized in *layers*, according to the standard International Organization for Standardization/Open System Interconnection (ISO/OSI) seven-layers model. Here are examples of performance indicators:

– at the Physical layer: bit error rate of a wireless link;

– at the Link/Medium Access Control (MAC) layer: collision probability in a common medium (e.g. wireless fidelity (WiFi)) and point-to-point delay;

– at the network layer: rejection rate in a router and sojourn time in a router;

– at the transport layer: number of Transmission Control Protocol (TCP) connections that can be simultaneously opened;

– at the application layer: response delay to a HyperText Transfer Protocol (HTTP) request.

The control of the performance of a network is achieved through the control of each of its components.

Before a user makes a decision on their choice of telecommunication facility (e.g. MultiProtocol Label Switching (MPLS) connection or WiFi local network), it is normal for them to want to know the parameters characterizing the transmission quality (delay, loss ratio, etc.) offered by this facility, in order to know whether this facility truly matches their needs. In other words, one must know the performance of this facility.

In case it is the user who has full responsibility for the telecommunications facility (typically a LAN), it is then up to them to assess the performance of the system.

When a user is dealing with a service provider, there will be a contractual relation, which is called in generic terms *service-level agreement* (SLA), between the QoS that the provider has to offer and the traffic that a user can submit. For instance, a delay less than 100 ms for traffic not exceeding 50 Mbps in average and 100 Mbps in peak rate.

1.2. Performance versus resources provisioning

The performance of a system depends strongly on the available resources. For example, a video broadcast of 2 Mbps cannot be achieved if the available bandwidth is only 1 Mbps. The performance of a system and the resource provisioning are two faces of the same problem, which is the matching between the requirement of an application and the resources that have to be committed. For instance, a video traffic with a sustainable rate of 5 Mbps and peak rate of 10 Mbps can certainly be supported by a dedicated bandwidth of 10 Mbps, the QoS will be perfectly guaranteed with zero loss ratio and no waiting delay. However, the dedicated 10 Mbps resource has to be paid. An *extremely* economic solution is to provide only 5 Mbps (the mean rate), with a buffer absorbing traffic submitted at higher instantaneous rates. The committed resources (5 Mbps and some random access memory (RAM) for buffering) should be less expensive than the dedicated bandwidth of 10 Mbps, but the QoS will be less good, with probable long delay and high packet loss ratio. An intermediary solution would be a bandwidth of $B \in (5, 10)$ Mbps with an adequate amount (L) of buffer. *Performance evaluation* aims to find a good couple of (B, L) for a given requirement (delay, loss ratio).

1.2.1. *Performance indicators*

The choice of relevant performance indicators intrinsically depends on the application. For example, a file transfer can allow a relatively large transmission delay, but claims lossless quality, whereas for videoconferencing, the standard end-to-end delay is 150 ms but it is tolerant to a limited amount of losses.

The performance of a computer network, or more generally the one of a network-based computer system, is characterized by a set of parameters:

– throughput;

– delay (end-to-end delay, delay at an intermediary node, etc.);

– load (utilization ratio);

– number of packets in the node (router, switch, etc.);

– loss ratio on a wireless link, rejection ratio in a router.

1.2.2. *Resources provisioning*

The resources involved in networking systems fall mainly in one of the following three categories:

– *processing*: it is usually the central processing unit (CPU) power, with impacts on the performance indicators such as number of packets routed by a router, number of HTTP requests processed by a Web server, etc.;

– *storage*: this can include both a volatile storage (RAM) or permanent storage (disk, flash memory). These resources act often as a buffer to absorb the random variations of the submitted work. Their provisioning affects indicators such as loss ratio in a router;

– *transmission*: from a modeling point of view, this resource plays a similar role to that of the processing capacity, with the particularity that, here, the "processing" consists of messages transmission (Internet Protocol (IP) packet, Ethernet frame, etc.). It affects indicators such as delay or loss ratio (considered jointly with the storage resource).

1.3. Methods of performance evaluation

Two classes of methods are generally used to assess the performance of a system: studies done directly on the system itself versus studies carried out through a model.

1.3.1. *Direct study*

This approach requires the *availability* of the physical system to be studied, or, at least, that of a conform copy. The advantage of a study conducted directly on the system itself (or a copy of it) is its intrinsic faithfulness to the reality. There is no distortion by the modeling process (see below) which,

by definition, cannot be an exact copy. However, this method is very expensive and often impossible:

– it is, for example, very difficult to stop an operational network to turn it into a test bed. The only reasonably feasible operations are measurements gathering;

– this study is by definition infeasible when conceiving a new system (which does not exist yet). We may note that it is precisely at this phase that there is a need to compare multiple design options for various possible scenarios.

There are indeed very few cases where we can carry out studies directly on physical systems, for reasons of availability, safety, etc. Therefore, in many cases, we have no choice but to try modeling. This book only considers approaches based on modeling.

1.3.2. *Modeling*

Failing to work directly on the system itself, it is necessary to work on a model. There are two main approaches:

– *physical modeling* in which we build a scale model (reduced model) of the system. This approach is often rather expensive. We will not deal with these kind of models and focus instead on abstract modeling, by using mathematical formalism and/or computer technology;

– *abstract modeling*: this modeling can be done in two ways:

- *mathematical model* (*analytical model*) that is carried out using mathematical frameworks such as the Queueing theory (or graph theory, etc.);

- *computer-based model* that is achieved by using appropriate programming language. In practice, we often use specialized software, for instance the `OMNeT++` simulation environment.

Model building requires an abstraction of the real system and often has to abandon some details that are considered as *not essential and thus negligible* with respect to the goal of the study. The usefulness of a modeling depends on the following two aspects:

– *relevance* of the abstraction, i.e. if the choice of neglecting certain details is justified;

– *faithfulness* to the selected components, i.e. if the latter is correctly modeled.

We will not deal directly with the issue of model building in general, i.e. the manner of choosing the details to be modeled. This choice depends primarily on the comprehension of the system and the goal of the study. Modeling is basically an art that one progressively learns to master through experience.

There is no general rule for assessing the quality of a model because it depends on the target of the study, which has a strong impact on the choice of abstraction and granularity of the modeling. It is worth noting that a model is usually not a true copy of the target system. Results from a modeling study should therefore always be examined with a critical spirit regarding their degree of representation.

1.4. Modeling

1.4.1. *Shortcomings*

Since we have chosen to work with modeling, we must first present its flaws so that readers are sufficiently aware of the precautions to take either in case they have to undertake a study by modeling or if they are simply users of results coming from a model.

A model, either analytical or computer-based, is a more or less simplified transcription of a real system to a virtual representation of the latter, by means of mathematical formalism or computer coding. There is therefore always a risk of lack of fidelity to the original. It may be due to an error of assessment in modeling. It can also be due to the fact that the tool used is not powerful enough to describe the system in detail, which is the case of mathematical modeling. In the case of computer modeling, although the tool itself is sufficiently powerful to reproduce in detail a computer system, it is often unwise to reproduce all the details and therefore there is still a need to fix the granularity by choosing the most relevant details. Moreover, a computer model is achieved through programming, so it may contain programming errors, so-called *bugs*.

1.4.2. *Advantages*

As aforementioned, it is often too expensive and/or simply not possible to directly work on the real system (neither even a copy nor a reduced model of it). Consequently, in spite of all the shortcomings and precautions to be taken, modeling remains the most accessible and effective means to study physical systems. The principal advantage of modeling lies in its force of parametrization and modularity, which makes it possible to generate various situations at will:

– it is by definition the only accessible approach during the design phase of a (future) system. This allows us to study and compare several options, without building physical prototypes;

– we can also build a model for a running operational system to carry out tests on the model in order to find a better adjustment of parameters of the running system without deteriorating the operation of the system itself;

– we can finally put the model of a system under various conditions, in order to locate its limits and to find remedies for undesired situations. A network operator can, for example, study different scenarios about the foreseeable needs of its (potential) users, in order to determine the best strategy for each scenario and the resources to be committed.

1.4.3. *Cost of modeling*

Modeling claims significant efforts in materials resources as well in human involvements.

Any modeling must begin with a good understanding of the target physical system. This one can be obtained, for an existing system, by means of long observation campaigns. The research of similar experiences constitutes a very useful alternative and/or complementary way. For a system in design phase, the observation campaign is replaced by a significant effort of documentary investigation. A good comprehension can be obtained by analysis of similar theoretical and/or case studies reported in the scientific and/or industrial literature.

Mathematical modeling is inexpensive in terms of material resource but claims the availability of high skill experts in mathematics. Moreover, it is not always guaranteed that we can establish a mathematical model that is faithful

enough to the real system. Most often, an analytical model is a quite approximative model.

Computer simulation requires, for the model building, a specialized programmer and long hours of development. The simulation phase requires adequate computational resources as well as long hours of human involvement devoted to the simulations run then results analysis. In addition, commercial simulation softwares are often rather expensive.

1.5. Types of modeling

We propose here a survey of different types of modeling for physical systems of various kinds and as a function of various investigation goals. A model can be:

– static or dynamic: depending on the situation, we can study the system at a specific point in time or to study its evolution in time. The first is called *static* modeling and the second *dynamic* modeling. The modeling of a computer network generally belongs to the category of dynamic modeling;

– stochastic or deterministic: depending on the existence or not of random elements, a model can be *deterministic* or *stochastic*. Computer network models are generally stochastic;

– continuous or discrete: in a *continuous* model, the state of the system evolves in a continuous way in time; whereas in a *discrete* model, the changes only take place punctually at a set of instants. Computer networks are discrete systems.

In conclusion, in this book we will deal mainly with *dynamic, stochastic* and *discrete* modeling.

1.6. Analytical modeling versus simulation

This book presents two methodological approaches of modeling, which are:

– *analytical modeling*, i.e. by establishing and solving mathematical models. These models are mainly *probabilistic* models. *Queueing theory* is among the most used methods;

– *computer-based simulation*, i.e. the building of a model by using a software tool and then by conducting *statistical* study of the system behavior through *virtual experiments* based on this model. The most suited approach for computer networks is the *discrete event simulation* (DES).

This book presents the Queueing theory and DES. These two tools are often jointly used for the modeling and the analysis of the computer systems for performance evaluation and/or resource dimensioning. They are two complementary approaches, both of which are often needed to conclude a study. Indeed:

– analytical modeling leads to firm and general results. Once we get an analytical model, if we want to study a new scenario, it suffices to take the adequate parameters of the target scenario and then recompute to get the new results. However, the constraints of mathematical tractability very often lead to a model which is too simplified compared to the reality;

– with simulation, the descriptive power provided by the programming tool makes it possible to model in a very detailed way the target system. However, to draw a conclusion with a minimum statistical quality, it is necessary to carry out a certain number of simulation campaigns. Moreover, the conclusion is valid only for a given situation. If we want to study a new scenario, it is necessary to run again simulation campaigns.

The two approaches are thus complementary: analytical modeling makes it possible to get a macroscopic outline of the problem, whereas simulation makes it possible to carry out more in-depth study on certain details. The link between two approaches is strengthened by the fact that simulations have to incorporate analytical modeling in certain parts of its implementation: it is indeed far from effective to reproduce all the behaviors of a system in the least amount of detail; certain parts can be replaced by their analytical models.

PART 1

Simulation

2

Introduction to Simulation

In this chapter we present generalities about discrete event simulation, as well as its relationship with mathematical modeling.

2.1. Presentation

Simulation is one of the most used tools for quantitative studies, both in academic research and in the industrial world. It helps to study the dynamic behavior of complex systems such as computer networks.

There are two families of simulation techniques, continuous simulation versus event-driven simulation (or discrete event simulation):

– *continuous simulation* studies the dynamic evolution of a system (physical, mechanical or chemical phenomena) which is described by parameters with continuous values. In fact, it is not really possible, with computer technology, to track the evolution of these parameters in a *continuous* way. Instead, the evolution is made by small steps (in space and in time). We can, for example, simulate the damage caused to the structure of a car following its collision against a wall;

– *discrete event simulation* (DES) is suitable for systems whose evolution takes place only punctually on a set of specific instants consecutively to the occurrence of some events. For example, the occupancy state of the buffer of a router changes only when the router receives or sends a packet. In this book, we deal solely with discrete event simulation (DES).

Performance Evaluation by Simulation and Analysis with Applications to Computer Networks, First Edition. Ken Chen. © ISTE Ltd 2015. Published by ISTE Ltd and John Wiley & Sons, Inc.

The purpose of a simulation is to reproduce, by software, a physical system (in this book, we speak mainly of computer networks) so that we can undertake statistical experiments on it. Through these experiments, we can evaluate the performance of the target system, as if we performed experiments on the real physical system. For example, we can check if it is possible to guarantee a certain quality of service (QoS) for a given level of traffic through a given path within Internet.

As aforementioned, we do not deal with the issue of model building itself. The manner of adequately choosing the salient details depends on each one's understanding of the system and the goal of the investigation. It is an art that one progressively learns to master. This book deals with the tools and techniques allowing the building of a computer-based model on one hand, and, on the other hand, its exploitation for statistical studies. There are three principal steps in a simulation study:

1) Modeling (partially using mathematical models) of the target system and its environment. Chapter 3 is devoted to the identification and the reproduction of the system and its environment with their respective probability distributions;

2) Computational building of the model, often by using a language dedicated to simulation. Chapter 4 presents some techniques which are specific to the programming of simulation models;

3) Statistical experiments. This involves correctly running simulation and gathering data, then making accurate analysis and interpretation of the data. Chapter 5 presents some statistical techniques for the control of simulation and the analysis of the data produced by simulations.

In Chapter 6, we provide an introduction to the `OMNeT++` simulation language through which we try to illustrate various theoretical concepts presented in this book. This chapter is also designed to help understand and make first steps with `OMNeT++`, an *open-source* generic simulation tool.

Among numerous books on simulation, the author likes to mention a very reduced sample [FIS 73, JAI 91, FIS 01, BAN 05, ASM 07]. The books [WEH 10, INC 07] provide some concrete applications to computer networks. Francophone readers can also consult [HEC 03, BEC 06].

2.2. Principle of discrete event simulation

2.2.1. *Evolution of a event-driven system*

Simulation aims to reproduce the in-time evolution of the target system. Discrete simulation applies to systems whose evolution can be described by a succession of *events*. Each event leads to some evolution of the *state* of the system; the evolution is thus *event-driven*. The *state*, say s, of a system is, by definition, the set of all of the parameters allowing to describe the system at this moment. The state of a system at a given instant t, $s(t)$, determines the system at that instant.

EXAMPLE 2.1.– We are interested in the filling of the buffer of a communication facility, say a router. The state of this system (buffer) at the instant t, $s(t)$, can be described by the number of customers (packets) in the buffer. $s(t)$ evolves under the joint effect of two processes:

– the arrival process (reception of a packet) which increases $s(t)$ by 1;

– the service process (transmission of a packet) which decreases $s(t)$ by 1.

For an *event* occurring at the moment t, we note by $s(t)^-$, the state of the system right before the event and $s(t)^+$, the one just after. We suppose that the initial state is $s(0) = 0$ and that the scheduling policy (the ordering rule) of the packets is First In First Out (FIFO). We present below a plausible scenario with its possible ramifications:

1) the arrival of one packet (denoted by p_1) at the moment t_1 is an event e_1 which changes the state into $s(t_1)^+ = 1$;

2) this event causes in its turn another event e_2 at $t_2 > t_1$ which is the end of the service (and the departure) of the packet p_1. By denoting the service duration of p_1 with S_1, t_2 is then perfectly determined with $t_2 = t_1 + S_1$. At t_2, e_2 (departure of p_1) takes places and leads the state $s(t)$ to $s(t_2)^+ = s(t_2)^- - 1$;

3) the exact value of $s(t_2)^-$, and thus $s(t_2)^+$, cannot be predicted at t_1. Indeed, between t_1 and t_2, there may be other arrivals, two of the plausible events are:

- no arrival, thus $s(t_2)^+ = 0$;

- one arrival at t_3 with $t_1 < t_3 < t_2$. This event is e_3 and the system's state becomes $s(t_3) = 2$;

4) in the second case (an arrival at t_3), there are two consequences:

- we must consider other possible arrivals between t_3 and t_2. Each new arrival increases the state by 1 assuming that the last such one occurs at t_l;

- the state $s(t_2)^+$ is thus unknown at t_3. We can simply confirm that $s(t_2)^+ = s(t_2)^- - 1$, where $s(t_2)^-$ is the same as the system's state right after the last event before e_2, i.e. $s(t_2)^- = s(t_l)^+$.

We see here that the system's evolution can be described if its current state is known. We also observe the two driving forces which make it evolve, namely the arrivals and the services.

In this sample scenario, the scheduling policy is assumed to be FIFO. If it is not the case, the evolution becomes more complex. In addition, the state of the system may not be able to be described solely by the number of the customers. Let us imagine a system with two classes (2 levels of priorities for example), the state of the system must be described, in this case, by (n_1, n_2), which respectively give the number of packets in each class.

Figure 2.1 yields a sample illustration produced by `OMNeT++` (see section 6.9). It concerns the first 10 events (denoted by `#1` to `#10` in the figure) of a $M/M/1$ queue with arrival rate $\lambda = 1/3$ s. and service rate $\mu = 1/4$ s. These events are chained as follows:

– the `source` generates the first customer at instant 0 (event 1), which is *immediately* sent to the (`fifo`) queue (event 2). Then, after having been serviced (event 3), the customer is sent to the `sink` (event 4 which follows the event 3) with no time lap;

– for the second customer, we can see a similar sequence of events with $(5 \rightarrow 6) \rightarrow (9 \rightarrow 10)$;

– during the service of the second customer, there has been arrival of the third customer. The latter has to wait (the events 7 and 8).

2.2.2. *Model programming*

The previous description of the evolution of the system suggests a possibility of carrying out simulations on computers. The two key-points are given in the next sections.

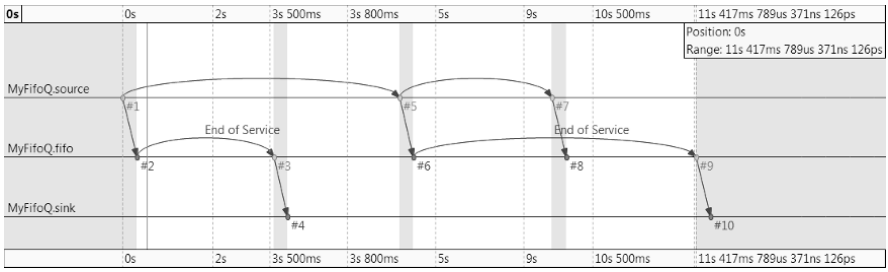

Figure 2.1. *Sequence of the first 10 events of an M/M/1 queue*

2.2.2.1. *Scheduler*

The evolution of an *event-driven* system will be seen through change of its state. This means that we can represent this evolution by a *chained list* called *Scheduler*. Each element in this chained list contains at least the identity of the event, the instant of its occurrence, as well as the action to be taken. This list will be sorted in a chronological way, through the occurrence instants of the events. In this way, the evolution of a system can be completely described. We thus obtain a means of simulating the chronological evolution of a system.

At any instant, the scheduler contains all of the events already known to occur at a given instant in the future. Thus, it is also referred to as *Future Events Set* (FES) or *Future Events List* (FEL).

2.2.2.2. *Object-oriented programming*

With the power of programming languages, we can achieve a description as detailed as possible of the behavior of the target system and its environment. Usually, it is based on object-oriented programming (OOP) languages, such as `Java` or `C++` because of their capacity in terms of modular design. Indeed, most of the physical systems are too complex to be modeled by a simple probability distribution. For an accurate description, we have to use OOP languages. The latter is particularly suitable for the modeling of computer networks. Indeed, computer networks consist of protocol entities (which are perfectly defined algorithmically) which are then assembled according to profiles (for example the IP/TCP family).

Within computational modeling, we can also adopt some mathematical models. Indeed, a detailed description is often more expensive in terms of simulation time than its counter-part based in a mathematical model. A

detailed description is only needed because we either really need to make an in-depth investigation focusing on the impacts of some details, or when we fail to establish a mathematical model with an acceptable level of fidelity to the physical system. For the components which do not require an in-depth examination, it is more effective to adopt an acceptable mathematical model, when it is possible.

2.3. Relationship with mathematical modeling

The analytical method and simulation both target the accurate modeling of the system. For this reason, they both need to observe them to identify the behavior of the system. However, an analytical model, due to limitations of mathematical tractability, provides only a more simplifying and consequently more restrictive model, whereas a simulation model can reproduce more details, as deeply as wished, and so is capable of providing a much more faithful model.

Under the adopted assumptions, an analytical model provides mathematically proved formulas. The latter is parametric, thus readily usable for investigating any scenario of interest. Each simulation campaign is, however, a singular experiment. In order to draw a statistically significant conclusion, we have to carry out a significant number of simulations. The same type of work must moreover be renewed each time we change one of the parameters of the system. We thus see the following duality:

– the analytical method requires the adoption of a model which is often some what too simplified. On the other hand, the results are exact. It is thus easy to examine various situations by varying parameters. The intellectual effort put aside, the mathematical modeling claims few physical resources;

– simulation makes it possible to obtain a more faithful modeling. However, obtaining the results is a long and delicate operation. Indeed, the more detailed the model is, the more the effort of programming is significant and the longer the time of simulation running will be. Moreover, each simulation campaign is a statistical experiment producing a result which is only valid for a given scenario. Its interpretation is subject to caution and may always suffer from a risk of uncertainty.

We then see the necessity to combine these two tools: the analytical tool for a macroscopic study which makes it possible to draw the broad outline, and the simulation tool for more targeted and more detailed study.

3

Modeling of Stochastic Behaviors

In this chapter, we examine at first the techniques allowing us to generate random variables (r.v.) that are uniformly distributed on the interval $(0, 1)$, from which we can generate r.v. of any other distribution. We then present some methods for generating r.v. of a given distribution from the r.v. of $U(0, 1)$, before presenting the use and generation of some commonly used distributions. We also give a short survey and some samples models related to computer networks. We will also briefly discuss how to find an adequate probability distribution for a given stochastic behavior, i.e. the issue of parameter estimation and hypothesis testing.

3.1. Introduction

Any study by simulation starts with the identification of the behavior of the system and its environment. These behaviors can be described by:

– an appropriate algorithm: for example, a packet with a certain destination address (`DA`) is routed to the interface `if0` according to the routing table;

– a probability distribution: for example, with a Bernoulli distribution, a packet is routed to the interface `if0` with probability p. This is a macroscopic modeling of the previous algorithmic example, if the focus is on the proportion rather than the matching of some particular destination;

– a combination of the two aforementioned approaches: for example, the size of the packets follows a Pareto distribution, the packets are routed according to their `DA` by consulting a routing table.

The descriptive and constructive power of the computer tool presents a fundamental advantage of simulation versus analytical modeling. Certain

Performance Evaluation by Simulation and Analysis with Applications to Computer Networks, First Edition. Ken Chen. © ISTE Ltd 2015. Published by ISTE Ltd and John Wiley & Sons, Inc.

phenomena or mechanisms can be described in an algorithmic manner, which allows their faithful reproduction by the computer tool. For example, we can perfectly reproduce a routing process of an Internet router by installing a true routing table in a simulation model of router and by introducing into each (simulated) packet a `DA` field with address under its authentic format.

However, even a simulation model cannot solely rely on algorithmic description and must resort to probabilistic modeling, sometimes by need, sometimes for reasons of effectiveness. Indeed, let us take again the aforementioned example of a routing module. If the goal of studies is not the precise destination of each packet but rather the macroscopic distribution of the traffics, such modeling would be too detailed and would consequently penalize the effectiveness of simulation. It is more judicious to replace the whole routing module by a simple probability distribution modeling the traffic's dispatching phenomenon. For example, if the router has two interfaces, we can simply use a Bernoulli distribution to separate the 2 subflows moving toward each of the two interfaces.

There are also situations where the physical object is too complex, or maybe even impossible, to be algorithmically described. It is, for example, the case of a source of traffic. Let us take again the example of a router. It is obvious that each incoming packet necessarily comes from a certain application running on a certain terminal. However, most of the time, it is not possible to describe completely the packet generation process. Indeed, the sending of a packet can be the result of a click on a uniform resource locator (URL) link which is itself a choice carried out in a completely hazardous way by a user, among, say, 20 URLs, on a Web page that one has randomly chosen. It would be impossible to describe algorithmically the generation of *each* packet. However, macroscopically, the packet generation phenomenon can likely be modeled by some probability distribution, the Poisson process is a good candidate in many cases.

In conclusion, we can say that the descriptive power of the simulation languages allows us to get a more faithful model than the analytical model. However, we must still very often resort to mathematical modeling to provide description of macroscopic stochastic behaviors of some components in a simulation model.

There are two determining factors helping to get an accurate modeling of a stochastic behavior:

1) *Estimation* of the probability distribution which models best the target behavior: many phenomena can be modeled effectively through an adequate probability distribution. For example, the number of phone calls, or the number of customers consulting a Website over a given period, can roughly be modeled by a Poisson distribution. It is a matter that is relevant to the statistics.

2) *Generation of sequences of random variables* to reproduce, artificially, the estimated distribution. It is about a fundamental aspect in simulation. Indeed, the quality of the simulation results depends directly on the quality of these r.v.The remainder of this chapter provides a detailed presentation of this topic.

3.2. Identification of stochastic behavior

The identification is generally achieved from the data produced by the target system model itself (or an equivalent) which can be obtained by field observations, through, for example, the traces (or log). It is an important branch of the statistics (see Chapter 13). We simply indicate the principal steps below.

The first phase is that of the data collection. It is a capital and often tedious task, because it claims many material resources and human involvement.

From these data, by graphic observation or by analysis, we can make an assumption about the candidate probability distribution. It is useful to draw a histogram or empirical cumulative distribution function (CDF) (see section 13.2.6) from the observed data, in order to compare this empirical distribution with usual distributions and help to identify a good candidate.

Let us imagine that our observations lead us to suppose that the number of visitors to a Website between 19 and 20 h has a Poisson distribution. This is an assumption that must be checked, i.e. the distribution must at first be identified to then be verified. The basic techniques are those presented in Chapter 13. There are two significant points:

1) *Parameters estimation*: most of the distributions are determined by a limited number of parameters. For example, a Poisson process is entirely determined by the parameter λ. There are some methods for the parameter

estimation of a probability distribution, in particular through the estimation of the sample mean and the sample variance (see section 13.3.3), or by using the method of maximum-likelihood estimation (MLE) (see section 13.3.4).

2) *Hypothesis testing*: to check the agreement (*goodness-of-fit*) between a set of data and its supposed distribution is a matter of testing the veracity of the hypothesis of the agreement between the data and the distribution (see section 13.4). We can use the Chi-square (χ^2) test or the Kolmogorov–Smirnov (K–S) test. If the hypothesis is rejected, another candidate distribution should be identified for testing.

There are also situations where the data are not available: it is obviously the case for a system in its design phase, it is also the case where the data are not accessible, for various reasons (technical, legal, etc.). In this case, we could set up a model either by analyzing the nature of the system, or by investigating similar cases (e.g. the number of users consulting a Website often has a Poisson distribution), we proceed often to a combination of both.

3.3. Generation of random variables

Once we get a good candidate distribution for a stochastic behavior (arrival, departure, etc.), we have to think about the generation of a sequence of r.v. according to this distribution. These r.v. will be used in simulation to actually *reproduce* the behavior. We will devote a large part of this chapter to deal with this problem, which is crucial to simulation.

The actual *faithfulness* and *randomness* of the generated r.v. occupy a central place in simulation. Indeed, it is through this way that a presumably random process is really created. If these r.v. do not reproduce the distribution in which it is supposed to represent, we can easily understand that the "results" produced by simulation will have little utility. Even worse, we could be misled by these erroneous results that we consider as good oracles.

The generation of r.v. is also one of the most frequent operations in a simulation. Indeed, to obtain a good statistical quality, it is often necessary to generate hundreds of thousands, even millions, of values during one simulation run. We thus have to pay attention to the *effectiveness* of the generation processes. This practical consideration also excludes use of the preestablished tables, because they are expensive to obtain, and often not sufficiently long.

A simulation is also a scientific experiment, which must be *reproducible*. This excludes the methods consisting of producing values at random, without any control or rule. A class of these processes is that which is based on independent physical phenomena, for example by taking the current time.

These considerations lead to an algorithmic generation of the random numbers. It is obvious that such a process suits well with the computer programming (automation of the generation) and thus easily reproducible. A corollary consequence related to this algorithmic generation is that the sequence of the numbers is not truly random. They are thus called *pseudorandom numbers* (PRNs), and their generators are called *pseudorandom number generator* (PRNG). Particular care must be taken on the quality of the generation, which must be submitted to various statistical tests.

3.4. Generation of $U(0,1)$ r.v.

The importance of the uniform r.v. on $(0,1)$ lies in the fact that it is theoretically possible to generate r.v. of any given distribution $\mathcal{L}$ from those of $U(0,1)$, if we know the inverse of the CDF of $\mathcal{L}$. Consequently, the only random variates having to be produced are r.v. according to $U(0,1)$. It is what we will show in what follows.

3.4.1. *Importance of* $U(0,1)$ *r.v.*

Given X, an r.v. with $F(.)$ as CDF. In order to simplify the discussion, we only consider the case where $F(.)$ is bijective. Denote $F^{-1}(.)$ as the inverse function of $F(.)$. For a given value x, let $u = F(x)$, thus $x = F^{-1}(u)$. $F(.)$ being a CDF, we have $u = F(x) \in [0,1]$.

Let $V \sim U(0,1)$. $Y = F^{-1}(V)$ is still an r.v. We are going to show that Y has the distribution $F(.)$; thus, it can be identified as X. Actually, the CDF of Y at the point y is $P[Y \leq y]$. We have:

$$P[Y \leq y] = P[F^{-1}(V) \leq y] = P[V \leq F(y)]$$

By remarking that $F(y)$ is a deterministic value (indeed, $F(.)$ is a known function and y a fixed value), we have:

$$P[Y \leq y] = P[V \leq F(y)] = F(y) \qquad [3.1]$$

As $F(.)$ is the distribution of X, we conclude that X and Y have the same distribution.

Due to this fundamental role of the $U(0,1)$ r.v., their generation must be treated with the greatest care, because it determines the quality of all the others r.v. deriving from it.

3.4.2. *Von Neumann's generator*

The generation of the $U(0,1)$ r.v. has been intensively studied. The encyclopedic work of Knuth [KNU 81] constitutes a reference. Nowadays, this topic is still an open research issue.

The methods that were proposed before computer era are either based on physical phenomena or on very long (and expensive) preestablished lists of random numbers. In addition to the high cost of these methods and lack of effectiveness and/or manipulability, they suffer from an intrinsic shortcoming: impossibility of reproducing an identical sequence. This is not compatible with the reproducibility principle of a scientific experiment.

Computers opened a new era. One of the first computer-oriented generators was proposed by Von Neumann in the 1940s that we will present here as an introduction to PRNG, because this one is very easy to understand and has already most of the common characteristics shared by other generators using more sophisticated algorithms.

One possible form of Von Neumann's method consists of starting with an integer n_0 of four digits ($0 < n_0 < 10,000$), for instance $n_0 = 4,321$. We then get n_1 which will be the four digits of the middle of n_0^2, and so on. In our example (see Table 3.1), $n_0^2 = 18,671,041$; thus, $n_1 = 6,710$. In the case of a number of odd number digits, we take one less digit on the right (the least significant) part. We obtain thus a sequence of n_i, $i \in \mathbb{N}$, between 0 and $10,000$. From them, we get $r_i = n_i/10,000$. This sequence is obviously distributed over $(0,1)$. If their distribution is uniform enough, we can

assimilate them as a sequence of uniform r.v. over $(0, 1)$, and we obtain in this way a generator of $U(0, 1)$.

i	n_i	r_i	n_i^2
0	4,321	0.4321	18,**671,0**41
1	6,710	0.6710	45,**024,1**00
2	241	0.2410	**58,08**1
3	5,808	0.5808	33,**732,8**64
4	7,328	0.7328	53,699,584

Table 3.1. *An example of PRNG with the PRNG of Von Neumann*

Far from being a perfect solution, this generator has already most of the characteristics of computer-based modern generators. We enumerate them as follows:

1) *Deterministic sequence*: it should be recognized that any sequence $\{n_i\}$ is a succession of perfectly deterministic numbers. They are created by a deterministic function $f(.)$ as well as an given initial number (n_0), which is called *seed*:

$$n_1 = f(n_0),\ n_2 = f(n_1),\ \ldots,\ n_{i+1} = f(n_i),\ \ldots$$

That is why these *algorithm-based* generators are called PRN.

2) *Role and choice of the seed*: the sequence is actually perfectly fixed by the *seed* n_0. Thus, seeds have to be chosen with precaution. Indeed, take the example of $n_0 = 1,000$, it is straightforward that the remaining sequence will be 0 starting from $n_1 = 0$.

3) *Periodicity*: as n_i is between 0 and $9,999$, so, for all i, there is surely a couple (i, P), with $P < 10,000$, such that $n_{P+i} = n_i$. From n_{i+P}, the sequence $\{n_i, \ldots, n_{i+P-1}\}$ will be *repeated*. Of course, the bigger P is, the better the approximation to a real uniform distribution over $(0, 10,000)$ will be.

4) *Quality*: we assimilate the $\{r_i = n_i/10,000\}$ as $U(0, 1)$ r.v., because we *assume implicitly* that the n_i are distributed uniformly over $(0, 10,000)$. This assumption has to be, of course, verified. On the other hand, in the best case, it would be a maximum of $9,999$ different values. This is not really the situation that would be produced by a real random process. The statistical quality of a PRNG must be checked. In the present case (four digits), it will not be a good PRNG.

3.4.3. *The LCG generators*

3.4.3.1. *Presentation*

As proposed in 1948, the *linear congruential generators* (LCGs) were the most popular generators until a recent date. So, they are still present in many simulation softwares. The generic form of the LCG is:

$$X_{n+1} = aX_n + b \mod M \qquad [3.2]$$

The $\{X_n\}$ being strictly less than M, the corresponding sequence of r.v. over $[0, 1)$, $\{r_n\}$, is obtained by the formula:

$$r_n = \frac{X_n}{M}. \qquad [3.3]$$

This sequence $\{X_n\}$ (and consequently, $\{r_n\}$) is perfectly determined by the *seed* X_0. An LCG is entirely characterized by (a, b, M). A sequence $\{X_i\}_{i\in\mathbb{N}}$ produced by the LCG (a, b, M) is completely determined by (a, b, M, X_0).

The formula [3.2] shows the following points, most of them have already been discovered through the example given in section 3.4.2:

– the numbers generated by the LCG are not really random: they form a sequence that is entirely determined by the seed X_0;

– there is surely a $P \leq M$ such that $X_P = X_0 \mod M$. After X_P, we restart the same sequence. P is the *period* of the generator;

– the choice of X_0 (the *seed)* is important, since it determines the entire sequence.

3.4.3.2. *Properties*

The period of an LCG is a very significant parameter. It is obvious that this one cannot exceed M. For certain values of seed, the period can even be largely lower than M. Hereafter, we give some results concerning the period of an LCG.

THEOREM 3.1.– An LCG gets its maximal period M under those conditions:

– b and M are relatively prime (coprime), i.e. the greatest common divisor (GCD) of b and M is 1;

– every prime divisor of M is also a divisor for $a-1$;

– if M is a multiple of 4 ($M = 4k$), so is $a-1$.

When $b = 0$, we speak about multiplicative LCG (MLCG). In the following, we focus on MLCG. The latter can be obtained more easily by computers than LCG with $b \neq 0$, which need an additional operation (sum). We report the following result.

THEOREM 3.2.– The maximal period of an MLCG is $M-1$. This period will be reached if and only if we get the following conditions:

– M is a prime number;

– a is a primitive root of M, which means:

$$\forall\, n,\ 1 \leq n \leq (M-2), \quad a^n \mod M \neq 1$$

There are thus two categories of generators MLCG following the nature of M:

1) M is a prime number: these generators will reach the maximum period;

2) M is a power of 2: these generators are easier to implement.

3.4.3.2.1. MLCG with $M = 2^k$

This category of MLCG is characterized by the following facts:

– the "modulo" operation is easy to realized by computers;

– the maximal period, $P = 2^{k-2} = M/4$, is reached if the following two conditions are satisfied:

- $a = 8i \pm 3$;

- X_0 is an *odd* number.

EXAMPLE 3.1.– Consider the MLCG defined by $X_{n+1} = 5X_n \mod 32$. We produce two sequences (with, respectively, $X_0 = 1$ and $X_0 = 2$) given as follows:

1) $X_0 = 1$: we have $\{1, 5, 25, 29, 17, 21, 9, 13, 1\}$. $X_8 = 1 = X_0$, thus $P = 8$;

2) $X_0 = 2$: we have $\{2, 10, 18, 26, 2\}$. $X_4 = 2 = X_0$, thus $P = 4$.

We also note that the first sequence contains only the odd numbers and the second one only the even numbers. This result illustrates once more that the generated numbers are not really uniformly distributed; otherwise, there would be at least a mixture of even and odd numbers.

3.4.3.2.2. MLCG with M primer number

We recall that in this case, the period can reach $M-1$ if a is such that:

$$\forall 1 \leq n \leq (M-2), \quad a^n \mod M \neq 1$$

EXAMPLE 3.2.– We consider the MLCG with $M = 31$. The possible values of a are $a \in \{3, 11, 12, 13, 17, 21, 22, 24\}$. Take $a = 3$. We get the generator:

$$X_{n+1} = 3X_n \mod 31$$

With $X_0 = 1$, we get the sequence $\{1, 3, 9, 27, 19, 26, 16, 17, 20, 29, 25, 13, 8, 24, 10, 30, 28, 22, 4, 12, 5, 15, 14, 11, 2, 6, 18, 23, 7, 21, 1\}$. The period is actually 31.

3.4.3.3. *Examples of LCG*

The parameters of an LCG must be selected in a very particular way so that the generated sequence of PRN really resembles a true $U(01)$ r.v. Prior to presenting some interesting LCG, here is an example (or rather a degenerated case) through which we can measure the impact of the choice of the parameters (a, b, M, X_0) on the quality of the LCG.

EXAMPLE 3.3.– We take $a = 1$ and $b = 1$; thus, $X_{n+1} = X_n + 1 \mod M$. Take $M = 4$ and $X_0 = 0$ for the sake of simplicity. We get the sequence $(0, 1, 2, 3, 0, 1, 2, 3, \dots)$. It is clear to see that the parameters of this LCG correspond perfectly to the aforementioned theorems, and it is nevertheless a very bad pseudorandom generator.

LCGs were largely adopted in various products and software tools. There are many proposals, with various qualities.

The MLCG defined by $a = 16,807 = 7^5$ and $M = 2,147,483,647 = 2^{31} - 1$, known as *minimal standard* LCG, deserves a particular mention, since it has very good statistical quality [PAR 88]. Its period is $2^{31} - 2$. An efficient algorithm for its implementation can also be found in [PAR 88]. This LCG was

used in many IBM systems. It is also available in OMNeT++, even if its role as the default generator has been replaced by the *Mersenne Twister* generator (see section 3.4.4.2).

As an example of bad LCG, we can quote the MLCG defined by $a = 65,539$ and $M = 2^{31}$. As known by Randu, it has been, in particular, implemented in IBM 370 series computers and was used in many studies. Studies showed thereafter its poor quality as PRNG [PAR 88].

3.4.4. *Advanced generators*

3.4.4.1. *Principle*

Studies have shown the existence of a certain correlation between the successive numbers generated by the LCG. We can note this phenomenon by tracing on a plane the points formed by (X_n, X_{n+1}). We will then see parallel lines appearing, which are more or less numerous as a function of the parameters of LCG. The existence of these lines shows that the distribution is not uniform.

Another concern which incites us to find other generators is that simulations increasingly claim longer and longer sequences. This could appear surprising at a first glance. Actually, with $M = 2^{31} - 1$, we can generate a long sequence of two billion unique numbers. However, the computing power which we have today and the complexity of the systems to be simulated means that these sequences can be exhausted. Indeed, let us take the simulation of an Asynchronous Transfer Mode (ATM) traffic, which is constituted by cells of 53 bytes (424 bits). A link at 600 Mbps is able to send about 1.5 million cells per second. One billion cells would be thus sent in approximately 660 sec, i.e. 12 min. Similar remarks could be made about the Internet routers that are able to process million of Internet Protocol (IP) packets per second.

For this reason, advanced generators were proposed. We indicate below two of the most used methods:

1) *Multiple recursive generator* (MRG) which takes the following generic form:

$$X_n = \sum_{i=1}^{K} a_i X_{n-i} \mod M$$

Such a generator requires K initial values.

2) *Combined LCG*: it consists of combining the sequences resulting from several LCG. Its generic form is as follows:

$$X_n = \sum_{i=1}^{K} (-1)^{i-1} Y_{n,i} \text{ Mod } (M_1 - 1)$$

where $Y_{n,i}$ is the nth value of the ith LCG and M_1 is the modulo of the first LCG. The output r.v. r_n is $r_n = X_n/M_1$ if $X_n > 0$ and $r_n = (M_1 - 1)/M_1$ if $X_n = 0$.

We present below more particularly, and very summarily, two modern generators, because of their performances and their popularity.

3.4.4.2. *Mersenne Twister generator*

One of the most powerful generators currently proposed is the *Mersenne Twister generator*. It uses the technique of twisted generalized shift register feedback (TGSFR). Very efficient and rapid implementations of this algorithms are available. In its most used variation (MT19937):

– its periodicity is $2^{19937} - 1$ (the numbers "$2^p - 1$" with p itself prime are primes referred as *Mersenne prime*);

– it can produce 623 independent sequences of 32 bits uniformly distributed numbers. This generator is adopted by many tools and languages (Python, the R language, etc.). It is the default generator of the `OMNeT++` simulation environment.

3.4.4.3. *L'Ecuyer's generator*

The generator that is proposed by Pierre L'Ecuyer is also a widely used one. It reaches a periodicity of about 2^{191} (about 3.1×10^{57}). The generator is implemented so that it can provide up to 1.8×10^{19} sub-sequences of independent $U(0, 1)$ r.v., with for each one a periodicity of 7.6×10^{22}. This generator is used in `NS-3`. In `OMNeT++`, there is also possibility of using it.

3.4.5. *Precaution and practice*

3.4.5.1. *Nature of the PRNG*

When we use a simulation language, it is useful to get information about generator(s) that the latter provides, by default and/or as an option, in order to know their characteristics and to measure their adequacy with simulations to be carried out.

If necessary, we could carry out tests for checking the quality of the generated r.v. It is to be pointed out, however, that the design and testing of pseudorandom generator claim advanced mathematical and statistical skills.

Ordinary users of a simulation tool seldom need these tests. On the other hand, it is very important to know the nature of the generators used by the simulator (e.g. the period) and precautions of use (e.g. choice of the seeds).

3.4.5.2. *Choice of seed*

The choice of the seed determines the sequence of generated numbers. It is thus a paramount parameter in a simulation. Here are some suggestions in this respect:

– it should be made sure that each simulation starts with a different seed. Otherwise, each simulation campaign that you believe "new" is actually simply a repetition of a previous run;

– for each simulation run, think of reporting the seed(s), so that the experiment can be reproduced for a possible posterior checking and/or replay. A simple and effective means is to put it in the same file where you record the results;

– get information about the existence of a possible list of the suggested seeds.

3.4.5.3. *Multiples substreams of RNG*

3.4.5.3.1. Principle

By default, there is only one PRNG that generates a single sequence of $U(01)$ r.v. from which all the r.v. of all the distributions used in a simulation run are derived. This could induce a risk of *artificial correlation* between simulated phenomena which, in reality, are independent. For example, the size of a packet and its arrival time are generally independent in real world. In simulation, if the size and the arrival time are both derived from the same basic $U(01)$ r.v.

sequence produced by the same PRNG, this creates an artificial link between these two quantities and could lead to a correlation which is artificial since it does not exist in the real world.

A current practice consists of generating the two phenomena (arrival, size) by two distinct PRNG. As generally only one algorithm of PRNG is used, people use, in practice, two sub-sequences of the implemented PRNG. It is preferable that these two sub-sequences do not overlap, in order to avoid any risk of correlation between them.

The *Mersenne Twister* generator is able to provide up to 623 independent sub-sequences. That of *l'Ecuyer* proposes 1.8×10^{19}.

3.4.5.3.2. Example of `OMNeT++`

Always from a practical point of view, not only do we need PRNG with sub-sequences capability, we also need implementations allowing their actual and effective exploration during simulations. Of course, it is always possible to program them in an *ad hoc* manner, but it is rather tedious and may cause *bugs*. `OMNeT++` provides a very interesting approach to handle the generation of independent sequences of random number. This approach (see example through listing 6.13 of section 6.9.3) is composed of two distinct methods that jointly provide the independence of various r.v. sequences:

1) Within each module, we can set distinct module-wide functional $U(01)$ r.v. generators. These generators are *virtual*, and they have to be *linked* to a real generator (see below). Their role is to allow functional separation of random variates generation. When we want to draw a value for a given r.v., we can specify the number of the basic $U(01)$ generator. By default, it is the generator 0 (`rng-0`). For instance, `uniform(1, 3, 6)` returns an r.v. $U(1, 3)$ by using the generator 6 (`rng-6`) of the related module.

2) At the system (simulator) level, we can set the number of real $U(0, 1)$ sub-sequences that will actually be used. These are generators producing really distinct, and independent, sequences. This is done through a configuration file (`.ini`), via `num-rngs` (default value is 1). We can then *link* each functional $U(0, 1)$ generator of each module to one of the system level real $U(0, 1)$ generator. For example, `M1.rng-3=7` associates the (functional) generator number 3 of module `M1` to simulator-level generator number 7.

This approach provides a very flexible, yet effective, manner to deal with various needs. Take again our example of arrival time and packet size. If we want to separate these two sequences, it suffices to associate the arrival time and the packet size generation to two generators (say 1 for arrival and 2 for size) in the same packet generator module. Then, at the simulation level, create two $U(0,1)$ generators and then associate them to, respectively, generator 1 and generator 2 of the packet generation module. We can easily adapt it to other situations with a simple configuration. For instance, if we decide to use only one simulator level $U(0,1)$ generator, it suffices to reconfigure generators 1 and 2 so that they are associated with the same simulator-level $U(0,1)$ generator.

3.4.5.4. *Quality of the PRNG*

The quality of a generator of PRNs is verifiable by statistical tests. Certain tests use common statistics tools, others are more specifically designed for PRNG.

It is basically a classical problem of statistics, namely the hypothesis testing, to assess the *goodness-of-fit* between the sequence produced by the generator and the assumed $U(0,1)$ uniform distribution. We can use statistical tests (see section 13.4) like the Chi-square (χ^2) test or the Kolmogorov–Smirnov test, we also can proceed to more advanced and/or specific analyses ([FIS 01, JAI 91]) such as spectral analysis or by drawing the couple (X_n, X_{n+1}).

3.5. Generation of a given distribution

We present below two methods for generating r.v. of any given distribution from the uniform r.v. on $(0,1)$. Accordingly, the distribution is known by its CDF for the *inverse transformation method* or by its probability density function (pdf) for the *acceptance–rejection method*. We also deals with the case of discrete r.v., as well as the composition and convolution of r.v.

3.5.1. *Inverse transformation method*

The approach used in section 3.4.1 indicated implicitly a method, which is called *inverse transformation method*, for the generation of r.v. of a given distribution, of which we know it CDF, $F(.)$, as well as its inverse $F^{-1}(.)$.

Let X be an r.v., with $F(.)$ its CDF and $F^{-1}(.)$ its inverse. This method showed (see equation [3.1]) that $F^{-1}(V)$, where $V \sim U(0,1)$, acts as an r.v. having the same distribution as that of X. It suffices to inverse the CDF of any distribution, then coupled it with a generator of $U(0,1)$ r.v., to obtain the r.v. of the target distribution.

EXAMPLE 3.4.– Let $X \sim Exp(\lambda)$. Its CDF is $F(x) = 1 - e^{-\lambda x}$ where $x \geq 0$. We get readily $F^{-1}(u) = -\frac{\log(1-u)}{\lambda}$. As u and $1-u$ are both r.v. of $U(0,1)$, the r.v. having $Exp(\lambda)$ are obtained with:

$$X = -\frac{\log(u)}{\lambda}$$

The main drawback of this method is we have to know the explicit form of $F(.)$, then to get $F^{-1}(.)$, and, eventually, be able to calculate $F^{-1}(r)$ for $r \in (0,1)$. The acceptance–rejection method offers an alternate way.

3.5.2. *Acceptance–rejection method*

The inverse transformation method requires the knowledge of the inverse of CDF. When this one is not easily invertible, we can resort to the *acceptance–rejection method*. In this case, we work with the pdf $f(x)$.

3.5.2.1. *Case of finite support*

We present at this step the method in the case where the pdf $f(x)$ is a bounded function with finite support, i.e.:

– $\exists\,(a,b)$ with $a < b$ such that $f(x) = 0$ for $x \notin [a,b]$;

– $\exists\, M > 0$ such that $\forall x, f(x) \leq M$.

We can imagine a rectangle $\mathcal{D} = [a,b] \times [0,M]$ covering the whole curve $f(x)$. The algorithm runs as follows:

1) generate u and v following $U(0,1)$;

2) set $x = a + (b-a)u$ and $y = Mv$;

3) if $y \leq f(x)$, then produce $X = x$ as the newly generated random outcome; otherwise, restart the algorithm.

Graphically speaking (see Figure 3.1), each run of the above algorithm produces one point defined by the couple $(x(u), y(v))$ which is *uniformly distributed* over the rectangle $\mathcal{D}$. The latter contains, by construction, the curve $f(x)$. If the point is under the curve, then we keep x as a value of X; otherwise, we have to generate a new point. In the example of the figure, $y_1 > f(x_1)$, the point (x_1, y_1) is not good; then we generate (x_2, y_2). As $y_2 < f(x_2)$, x_2 is presented as a newly generated value following the given distribution.

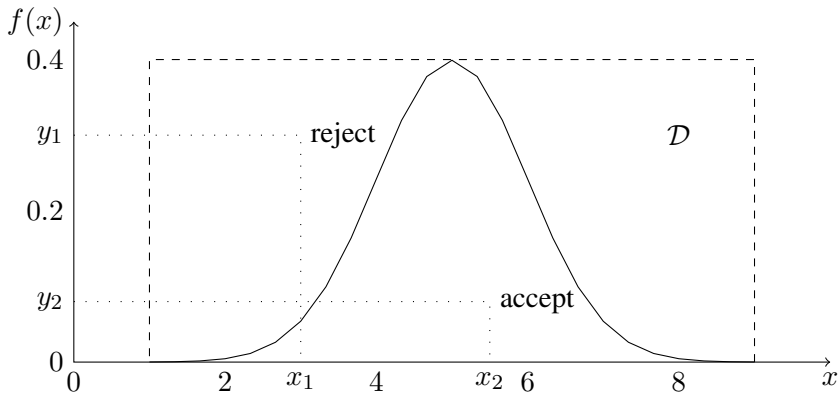

Figure 3.1. *Generation of r.v. by the acceptance-rejection method*

It is easy to see that the generated r.v. has the distribution $f(x)$. Indeed, for every $l \in (a, b)$, $P[X \leq l]$ is, by construction of this method, equal to the surface under the curve $f(x)$ between a and l, we have thus:

$$P[X \leq l] = \int_a^l f(u)du = F(l)$$

X follows actually the distribution $f(x)$.

The major drawback of this method is that we have to carry out quite a number of trials for a single outcome. It can be seen, by the nature of this method, that the proportion of useful trials (those do produce a random number) is the same as the ratio between the area under the pdf $f(.)$ and the surface of $\mathcal{D}$.

3.5.2.2. *Generalized version*

The generalized version allows us to take into account distributions with an unbounded support. The only condition is the pdf $f(.)$ of the target distribution being bounded by the one (denoted as $g(.)$) of another distribution and we are capable of generating r.v. of the latter distribution. It is possible to use an amplifier $M > 0$ to ensure

$$\forall x, f(x) \leq Mg(x)$$

The process runs as follows:

– generate a value of x according to the distribution $g(x)$;

– generate a value of y which is uniformly distributed over $(0, Mg(x))$;

– if $y \leq f(x)$, then keep x as a new random variate following $f(x)$; otherwise, restart.

A typical application is the generation of a distribution having complex pdf, but the latter can be bounded by the pdf of another distribution which is easier to generate. We can consider the basic method presented in the previous section as a particular case of the general method presented here, by taking for $g(x)$ the one of the uniform distribution $U(a, b)$ and $M = \max\{(b-a)f(x),\ x \in [a, b]\}$.

3.5.3. *Generation of discrete r.v.*

3.5.3.1. *Case of the finite discrete r.v.*

There is a generic method to generate discrete r.v. with a finite number of values, by directly applying the inverse transformation method. Here is its principle.

Take a discrete r.v. $X \in \{a_i\}_{i=1...N}$, with distribution $\{p_i = P[X = a_i]\}_{i=1...N}$. We know that $\sum_{i=1}^{N} p_i = 1$. We can also note that the interval $(0, 1)$ can be divided into N contiguous subintervals I_i of respective width $p_1, p_2, \ldots, p_N$, with a bijection between a_i and I_i.

The next step consists of generating an r.v. u according to $U(0, 1)$, then locating the corresponding interval that contains u, say I_k. We can then produce a_k as the newly generated outcome of X.

EXAMPLE 3.5.– The simplest case is a Bernoulli distribution $X \sim B(p)$, with $p = P[X = 1]$. Thus, $I_0 = [0, 1 - p)$ and $I_1 = [1 - p, 1]$. Any $u \geq 1 - p$ provides $X = 1$, and if $u < 1 - p$, then $X = 0$.

This method is generic. It depends only on the distribution of X, i.e. the set of $\{p_i\}_{i=1...,N}$. From computational point of view, the operation is a kind of sorting: it is about to find the right index k among N possible values. The operation could become time-consuming for large N.

3.5.3.2. *Case of countably infinite discrete r.v.*

If X is a countably infinite r.v., N tends to ∞ and the aforementioned method can no longer be used. There is often no need to study individually each value beyond a certain threshold S. We could then gather the cases $\{S, S + 1, \ldots, \infty\}$ into a virtually single case. We can then apply the method of the previous section. However, this is only effective if S is not too big.

For certain discrete variables, specific methods are developed to deal with the infinite support. We present cases of the geometric distribution and the Poisson distribution in the following section.

3.5.4. *Particular case*

Stochastic phenomena that we can meet are sometimes very complex and cannot be described by a common distribution such as, for example, the exponential distribution or the normal distribution. We can, of course, always apply the acceptance–rejection method if we find a dominating distribution of which we are capable of generating random numbers.

Some of these situations can, however, be identified (or approximated) as the composition or the convolution of some more elementary distributions. In these cases, it is also possible to apply the inverse transformation method that is more effective.

3.5.4.1. *Composition*

This corresponds to the case where an r.v. X (with its CDF $F_X(.)$) can be considered as being constituted from n other r.v. X_i (with, respectively, their CDF $F_{X_i}(.)$) proportionally with weight p_i, i.e.

$$F_X(x) = \sum_{i=1}^{n} p_i F_{X_i}(x), \text{ with } \sum_{i=1}^{n} p_i = 1$$

In this case, the generation of X can be done in two steps:

1) generate $u \sim U(0,1)$ in order to select the elementary CDF, k, to be used for the current generation of X, $k \in \{1, \ldots, n\}$;

2) generate X as an r.v. according to $F_k(.)$.

3.5.4.2. *Convolution*

This is the case of an r.v. X which can be considered as the sum of n other r.v. X_i:

$$X = \sum_{i=1}^{n} X_i$$

In this case, the generation of X is achieved first by the generation of an outcome for each of the n X_i then by summing them together.

3.6. Some commonly used distributions and their generation

In this section, we present some usual distributions, with a computer-network-oriented interpretation. For each of them, we specify use cases, the meaning and getting of their parameters, as well as the generation process. The definition and the principal characteristics of these distributions are not recalled, since they can be found in section 12.5.

Unless otherwise specified, u and v indicate auxiliary r.v. with distribution $U(0,1)$ (those given by the PRNG), $f(.)$ indicates the pdf of the target distribution and $F(.)$ its CDF.

3.6.1. *Uniform distribution*

3.6.1.1. *Utilization*

The uniform distribution $U(a, b)$ (see section 12.5.5) is the simplest model one could use to model a continuous random phenomenon.

It is used in particular if the randomness of a component is established with its variation range, whereas we have no more precise knowledge on its stochastic behavior other than the fact that there is no evidence of dominant values (mode).

3.6.1.2. *Parameters*

The $U(a, b)$ distribution, with $a < b$, is fixed by two parameters that correspond to the two extremities of the variation range of the modeled physical quantity.

3.6.1.3. *Generation*

The $U(a, b)$ distribution, with $a < b$, is generated with a simple recalibrating operation of an r.v. $U(0, 1)$. Let $X \sim U(a, b), a < b$, X is given by:

$$X = a + (b - a)u$$

3.6.2. *Triangular distribution*

3.6.2.1. *Utilization*

The triangular distribution $T(a, b, c)$ (see section 12.5.6) allows us to model a random phenomenon of which we know the variation range as well as its dominant point. This often corresponds to situations where we have few data (physical or logistical difficult to carry data collection). For example, if a transmission delay ranges between 10 and 30 ms, with 25 ms being the most frequent one, the phenomenon can be modeled by $T(10, 25, 30)$.

3.6.2.2. *Parameters*

The distribution $T(a, b, c)$, with $a < b < c$, has three parameters of which two (a, c) correspond to the two ends of the variation of the physical quantity. The parameter b is the mode (see section 13.2.5) of the variable, i.e. the most frequent value.

3.6.2.3. *Generation*

The generation of $X \sim T(a,b,c)$ is done as follows, with two possible cases according to the value of u:

$$X = \begin{cases} a + \sqrt{(b-a)(c-a)u} & 0 \leq u \leq \frac{b-a}{c-a} \\ b + \sqrt{[(c-a)u - (b-a)](c-b)} & \frac{b-a}{c-a} < u \leq 1 \end{cases}$$

3.6.3. *Exponential distribution*

3.6.3.1. *Utilization*

The exponential distribution $Exp(\lambda)$ (see section 12.5.7) is one of the most used distributions in simulation of computer networks.

3.6.3.1.1. Arrival process

It is in particular used to generate a Poisson arrival process. Indeed, it is established (see section 8.2) that a Poisson process with parameter λ is the one whose interarrival intervals have an exponential distribution of the same parameter λ. Let t_i be the instant at which the ith customer arrives, the instant of the next customer, the $(i+1)$th one, is given by $t_{i+1} = t_i + X_i$ where $X_i \sim \text{Exp}(\lambda)$.

The number of customers consulting a Website or the phone calls during a given period are examples of Poisson process.

REMARK 3.1.– If we want to generate a *fixed* number, say N, of Poisson arrivals during a given interval of duration T, we can also use the fact (see section 8.6) that these N arrivals are *uniformly* distributed over the interval. More generally, this *could* lead to an alternative way to generate Poisson arrivals, which is given blow (for the sake of simplicity, we work on the interval $[0,T]$):

1) Generate the (random) number (N) of arrivals according to the Poisson distribution with parameter λT (see section 3.6.10.3).

2) Generate N uniformly distributed instants (see section 3.6.1.3) over $[0,T]$, noted $t_1, \ldots, t_N$.

3) Sort these N instants in chronological order $t_1 < \ldots < t_N$.

This approach claims, however, *more operations* than the previous one.

3.6.3.1.2. Memoryless phenomena

More generally, the exponential distribution is used to model any *memoryless* phenomenon.

3.6.3.2. *Parameter*

The distribution $Exp(\lambda)$ is determined by only one parameter λ, which is the inverse of the mean value of the distribution. One practical way to obtain λ of a physical quantity $X \sim Exp(\lambda)$ consists of obtaining an estimation of $E[X]$, via, for instance, the *sample mean* estimator $\overline{X}$ (see equation [13.3]). We then get $\lambda = \frac{1}{\overline{X}}$.

Physically, λ can also be considered as an arrival rate, i.e. the mean number of arrivals during a time unit. This yields another way to estimate this parameter.

3.6.3.3. *Generation*

Let $X \sim Exp(\lambda)$. X can be generated (see example 3.4 of section 3.5.1) by:

$$X = -\frac{\log u}{\lambda}$$

3.6.4. *Pareto distribution*

3.6.4.1. *Utilization*

The Pareto distribution with parameters (s, α) (see section 12.5.10) is used to model many phenomena which have the property of "long tail", i.e. the large values portion of the variable has a non-negligible part among all the possible values. This distribution can be used to model packet size in the Internet and the number of URL references pointing toward a single Web page, etc.

3.6.4.2. *Parameters*

The parameter s corresponds to the threshold from which the variable is defined. It can be imposed by the physical characteristics. Otherwise, s can be estimated from the sample data $\{x_i\}_{i=1,\ldots n}$ with:

$$s = \min\{x_i\}_{i=1,\ldots n}$$

The parameter α can be deduced from the same sample $\{x_i\}_{i=1,...n}$ by using the method of *maximum likelihood estimation* (MLE, see section 13.3.4). We get α by the following formula:

$$\alpha = \frac{n}{\sum_{i=1}^{n} \log(x_i/s)} \qquad [3.4]$$

We can also note that the complement of the CDF of a Pareto distribution has the generic form $P[X \geq x] \sim cx^{-\alpha}$. Thus:

$$\log(P[X \geq x]) = \log\Big(1 - F(x)\Big) = -\alpha \log x + B$$

where B is a constant, i.e. α is the slope of the curve $1 - F(x)$ in scale log-log. This indicates another way for getting α.

3.6.4.3. *Generation*

Random variates of Pareto distribution with parameters (s, α) can be generated by:

$$X = \frac{s}{u^{1/\alpha}}$$

3.6.5. *Normal distribution*

3.6.5.1. *Utilization*

The normal distribution $N(\mu, \sigma^2)$ (see section 12.5.8) is one of the most fundamental distributions. It can be used for the modeling of white noise, or, more generally any cumulative impact of a large number of independent and identically distributed (i.i.d.) r.v.

3.6.5.2. *Parameters*

The parameters μ and σ^2 correspond, respectively, to the mean and the variance. They can be obtained via the sample mean estimator (see equation [13.3]) and the sample variance estimator (see equation [13.4]).

3.6.5.3. *Generation*

We obtain simultaneously a couple of r.v. (X, Y) having, respectively, the normal distributions $N(\mu_x, \sigma_x^2)$ and $N(\mu_y, \sigma_y^2)$ by the formula:

$$\begin{cases} X = \mu_x + \sigma_x \cos(2\pi u)\sqrt{-2\log v} \\ Y = \mu_y + \sigma_y \sin(2\pi u)\sqrt{-2\log v} \end{cases}$$

3.6.6. *Log-normal distribution*

3.6.6.1. *Utilization*

A log-normal variable (see section 12.5.9) is a strictly positive r.v. whose logarithm has a normal distribution. The log-normal distribution can be used to model the duration of a session or the size of a file. Studies [MIT 04] showed that the log-normal distribution and the Pareto distribution share similar fields of applications.

3.6.6.2. *Parameters*

The log-normal distribution is defined by two parameters: $LN(\mu, \sigma^2)$, with:

$$X \sim LN(\mu, \sigma^2) \Longleftrightarrow \ln X \sim N(\mu, \sigma^2)$$

They are obtained through the corresponding normal distribution as aforementioned.

3.6.6.3. *Generation*

The generation of a log-normal r.v. is obtained by taking $Y = e^X$ where X is an r.v. generated according to the corresponding normal distribution. As we generate X per pair, the variables of a log-normal distribution are also generated per pair.

3.6.7. *Bernoulli distribution*

3.6.7.1. *Utilization*

The Bernoulli distribution $B(p)$ (see section 12.5.1) provides a simple model for any binary phenomenon in which we know the respective proportions: transmission error, loss of a packet, routing of a packet toward one particular networking interface, etc.

3.6.7.2. *Parameter*

The distribution $B(p)$ is defined by only one parameter p that can be obtained by measuring the frequency of one of the two values.

3.6.7.3. *Generation*

Let $X \sim B(p)$, with $P[X = 1] = p$. It suffices to simply generate an r.v. following $U(0, 1)$. If $u < 1-p$, then $X = 0$ otherwise $X = 1$, or, alternatively, if $u < p$, then $X = 1$ otherwise $X = 0$.

The preceding method requires a comparison operation. There is also an alternative method that is carried out in the following way:

$$X = \lfloor p + u \rfloor$$

where $\lfloor r \rfloor$ is the ceiling function that gives the integer immediately lower than or equal to r. Indeed, it is in a proportion of $1 - p$ that we have $u < 1 - p$. This method also leads to $P(X = 0) = 1 - p$ and $P(X = 1) = p$.

3.6.8. *Binomial distribution*

3.6.8.1. *Utilization*

The binomial distribution $B(n, p)$ (see section 12.5.2) can be interpreted as the number of positive binary tests among a total of n tests carried out independently and under the same conditions, with a probability of success which is of worth p.

3.6.8.2. *Parameters*

The distribution $B(n, p)$ is determined by two parameters: i) n which is the total number of tests and ii) p the probability of success for each test.

3.6.8.3. *Generation*

Let $X \sim B(n, p)$. X can be considered as the sum of n r.v. $Y_i \sim B(p)$. Thus, it suffices to generate n r.v. $Y_i \sim B(p)$ for $i = 1 \ldots n$, their sum gives X:

$$X = \sum_{i=1}^{n} Y_i$$

3.6.9. *Geometric distribution*

3.6.9.1. *Utilization*

The geometric distribution $Geo(p)$ (see section 12.5.4) allows us to model any repetitive phenomenon until the satisfaction of a criterion. It can, in particular, be used to model the retransmissions of a packet until its good reception.

3.6.9.2. *Parameter*

The geometric distribution $Geo(p)$ is determined by only one parameter which corresponds to the probability of failure.

3.6.9.3. *Generation*

The r.v. $X \sim Geo(p)$, with $P[X = i] = p^i(1-p)$ for $i \in \mathbb{N}$, has an infinity of possible values, the generic method for generation of the discrete r.v. (see section 3.5.3) does not apply to it and it needs *ad hoc* methods

One of the methods consists of simulating the underlying phenomenon by drawing u as much as needed until we get $u > p$, the number of draw minus 1 will then be the generated outcome.

Another method is deduced from a relation between the exponential distribution and the geometric distribution. Given $Y \sim Exp(\lambda)$. For $k \in \mathbb{N}$:

$$P[k \leq Y < k+1] = \lambda \int_k^{k+1} e^{-\lambda u} du = e^{-\lambda k}(1 - e^{-\lambda})$$

This quantity is equal to $P[X = k]$ for $X \sim Geo(e^{-\lambda})$, i.e. with $p = e^{-\lambda}$. So, $\log p = -\lambda$. The r.v. Y is generated with the formula $Y = -\frac{\log u}{\lambda}$. On the other hand, we have proven that:

$$P[k \leq Y < k+1] = P[X = k]$$

Thus, the generation of X is given by:

$$X = \lfloor \frac{\log u}{\log p} \rfloor$$

where $\lfloor r \rfloor$ is the ceiling function giving the integer immediately lower than or equal to r, as aforementioned.

3.6.10. *Poisson distribution*

3.6.10.1. *Utilization*

The Poisson distribution $P(\lambda)$ (see section 12.5.3) allows us to model various *memoryless* phenomena, in particular various arrivals phenomena.

3.6.10.2. *Parameter*

The Poisson distribution is determined by only one parameter λ which is the mean of the variable.

3.6.10.3. *Generation*

Let $X \sim P(\lambda)$. It is once again an r.v. with an infinity of possible values. One method of generation consists of running the following algorithm:

1) `initialization with` $L = \exp(-\lambda), t = u, X = 0$;

2) `While` $\{t > L\}$ `Do`:

- $t := t \times u$;

- $X := X + 1$;

3) `end While`;

4) `yield` X `as the generated r.v.`

This algorithm corresponds to the formula:

$$X = \min\{n \ / \ (\Pi_{j=0}^{n} u_j) \leq e^{-\lambda}\}$$

3.7. Applications to computer networks

We present below some studies and models related to systems and phenomena in the computer networks, as illustration of the general principles presented in this chapter, and also as examples of application of them.

In [FLO 01], which is one of the pioneer studies on the modeling of various phenomena in the Internet, the following considerations have been presented:

1) the traffic is not so stationary: we note, in particular, a pattern change as a function of the time within a day, as well as the day within a week, according to the human activities. Moreover, certain traffics are generated by automatic processes;

2) correlation: there is in the long run a non-negligible correlation of the consolidated traffic (in terms of aggregation of the packets coming from various sources). This means that the consolidated traffic cannot be modeled by a Poisson process. The latter is better suited for the modeling of *memoryless* phenomena without long-term correlation. The long-term dependence appears in particular through the following observations: the traffic under various time scales (millisecond, second, minutes, etc.) has similar forms. This is referred to as *self-similarity*;

3) the Poisson process is nevertheless still largely present. It can in particular be used to model the birth of various types of sessions, i.e. the starting of an activity. Here are some examples of session: a document transfer through File Transfer Protocol (FTP), visit of a Website, remote access to a distant site, etc.;

4) the duration of a session can often be modeled by a log-normal distribution, even though numerous exceptions exist;

5) a significant number of physical quantities show the "long tail" phenomenon, i.e. large values represent a non-negligible share, or, in other words, the variance of the quantity is significant (with tendency toward the infinity). Among these quantities are in particular the size of the files (in particular those under UNIX), the number of Web pages, the volume of a file transfer, etc. These phenomena can be modeled by the Pareto distribution (see Figure 12.4), with $\alpha < 2$ (which leads to an infinite variance). The log-normal distribution is also a good candidate, even if it captures less good behavior at the high end (the one of the greatest values).

The Markov models (see Chapter 14) constitute a powerful tool for the modeling of many phenomena. The Markov model with two states (see 14.2.3), which is known as Gilbert–Elliot model or ON/OFF model, can be used as a basis for modeling non-homogeneous disturbances with two loss rates (high and low), or the telephone conversations that are composed of a succession of alternations between the period of speech and that of silence, etc.

Often, the phenomenon under study is too complex to be modeled by only one distribution. We can adopt a hierarchical modeling (see [CHO 99], one of the first significant works on Web traffic modeling). For example, there is a consensus to model the traffic related to the Website consultations in several levels:

– session level: the arrival of a new visitor, which starts their session of consultation, is generally modeled by a Poisson process;

– during a session: there are several proposals with different approaches. There are two main parameters (and view points):

- number of consulted pages during the whole session (with various models such as geometric, Weibull, log-normal, etc.);

- duration between two successive consultations (with various models such as geometric, Weibull, Pareto, etc.);

- this gives already a first ramification of the possible models that could be suggested. Moreover, certain models introduce a third dimension by cutting out a session into subsessions. Certain models also consider a fourth dimension with the consultations modes (fast research, in-depth reading, etc.);

– modeling of a page: a Web page being composed of various objects (text, images, etc.), various models are also proposed to describe the number of objects as well as their sizes;

– mode of transmission: we should not forget the final goal of this modeling which is about to model the generated traffic. This latter depends on the mode of connection in Transmission Control Protocol (TCP) and is thus related to the model of TCP traffic.

It is not the intention of the author to present in a precise way these models for Web traffic. In addition, these models have to evolve because various factors, technologies of Web page composition, page content, user's behavior, etc., are in constant evolution. The goal here is to show the complexity of the modeling of the phenomena in the Internet, and to show, at the same time, that it is possible to model them in a modular and hierarchical way. The author wants to point out that in case we fail to find a satisfactory probabilistic model, we can always think about algorithmic description or trace-based replay.

In a modeling, it is also necessary to take account of the technological constraints. Let us take the example of the modeling of traffic issued from a telephone conversation. If the vocal activities can be modeled by a Markov

process with two states[1], the traffic generation during the active periods is done by the coders which have to respect standards. For example, an usual combination associating the standard G.722 and Real-time Transport Protocol (RTP) (profile 9, RFC3551) stipulates that every 20 ms, we generate a packet (of 160 bytes) to constitute a flow of 64 Kbps.

As a conclusion, we suggest the following practice when we need to build a model for some phenomenon in the Internet:

– frequent phenomena (Web activity, file transfer, audio, video, etc.) have already been studied intensively. We have to proceed, first of all, to identify an existing model which is close enough to the target phenomenon;

– the existence of many studies shows that there is no one universal model. Moreover, the technological and usage evolutions are very fast in the Internet, it is also necessary to check the conditions under which a model was developed, in order to assess its real adequacy with the target phenomenon. There is often a need to adapt an existing model;

– certain models have rather complex mathematical formulation. If, from the view point of mathematical tractability, this complexity could arise problem for a complete analytical solution; in terms of simulation on the contrary, their reproduction (i.e. generation of the r.v.) generally has no major difficulty.

1 Note that more elaborate models in three states (or more) also exist by modeling a conversation with more details.

4

Simulation Languages

In this chapter we deal with the general aspects of the simulation languages: its characteristics and specific features. We then discuss the debugging of a simulation model.

4.1. Simulation languages

4.1.1. *Presentation*

Simulations are generally done by means of suitable programming tool. We will refer them as *simulation language*, whereas some of them are simply a particular library of a general-purpose language, and others are provided with a specialized integrated developments environment (IDE) with graphic interface and animation.

There is a great number of simulation languages[1]. Many of them are based on a object-oriented programming (OOP) language, to which we add some auxiliary tools which are specific to simulation, like PRNG, data-collection, model configuration or simulation run handling. In addition to these functional extensions, we can find some utility tools such as statistical data analysis tool, graphic interface, etc. Thus, there is a proliferation of simulation languages with different combination of these elements. In addition, simulation languages are often designed for a target application domain, with specialized libraries (pre-established models) for the latter. An

1 Personally, the author has used `SimScript`, `QNAP2`, `NS2`, `OPNET`, `OMNeT++`, as well as some more specialized tools.

Performance Evaluation by Simulation and Analysis with Applications to Computer Networks, First Edition. Ken Chen. © ISTE Ltd 2015. Published by ISTE Ltd and John Wiley & Sons, Inc.

example is the NS2 which completely lacks decent analysis tools, but is provided with facilities to configure and obtain a trace-file as complete and detailed as it wishes; it also provides a rich collection of models for various Internet protocols.

4.1.2. *Main programming features*

As aforementioned, the key-element of discrete event simulations is the handling of events. The basic part of any simulation language is thus the *scheduler* (see section 4.2), which makes the simulated system evolve in simulated time.

Another property of simulation (versus analytical modeling) is its capacity to model complex systems. An *OOP* language is well suited for such a construction, with a heritage of the existing objects, to build a system increasingly more complex by means of the heritage of the existing blocks. This functionality is particularly important for simulation of computer networks.

Computer networks are distributed systems. They are composed of independent and interacting entities. The interactions are carried out via exchanges of messages of various formats, under the form of Protocol Data Unit (PDU). An OOP language is also well suited for message exchange. Indeed, from the view point of OOP, messages of a given format are objects belonging to a particular object class.

4.1.3. *Choice of a simulation language*

The choice of a simulation language depends on several factors which we will develop below.

First of all, it is of course the adequacy of the tool with the system under study. Computer networks are very complex distributed systems. It is thus preferable that the programming language is object-oriented.

The languages are often developed for a given application context, with models for main functional entities of the context. For instance, OMNeT++ provides a whole set of modules (the environment inet) modeling principal

entities of the Internet; there are also other packages such as the VEINS package which is specialized in the modeling of the vehicular networks. The existence of such collections of pre-established models is an element also to be taken into account, because it considerably reduces the cost of model building, by exploiting the already existing entities, rather than "reinventing the wheel".

The criterion of the "community" is also very important, especially in the case of academic research related to some popular problems. It is indeed useful that the simulation context (protocol profile, basic mechanisms, etc.) involves common basic elements as much as possible, in order to make the comparison of the results with other investigations of the same community more relevant and convincing.

The facility of learning is also an element of consideration. This facility includes the learning of the language itself as well as the mastering of its library modules. A powerful language is often also a language which claims a long learning cycle. If we fail to well understand the language and/or its pre-established models, this could lead to modeling errors which are not always easily discovered.

The financial cost, in terms of licence as well as in terms of human involvement, is also a factor to be considered.

In this book, we chose to present the OMNeT++ language which will be covered in Chapter 6. Indeed:

– it is a language which, due to its design and the offered functionalities, offers a good framework for illustrating general concepts and practices of simulation;

– it is based on C++ language which is object-oriented. The C++ language is a widespread language and largely taught;

– in addition, the OMNeT++ framework comes with quite a lot of packages and modules especially developed for the computer networks;

– it is *open source*. As stated by OMNeT++, its license "provides non-commercial users of OMNeT++ with rights that are similar to the well-known GNU General Public License".

4.2. Scheduler

The scheduler primarily contains a list of events (called Future Events Set (FES) or Future Events List (FEL)). Each event is identified by the moment at which it takes place, the entity on which it takes place, and, of course, the action (process) to be realized. It is possible to have several events simultaneously. The various operations of a scheduler over the list of events are as follows:

– pick-up of the *next event* (head of the list) and the launching of its action;

– insertion of a (future) event, this operation is always driven by the current event: an arrival leads to the scheduling of the next arrival, the beginning of a transmission leads to the setting of its end, etc.;

– withdrawal of an event: in addition to the withdrawal of the current event at its termination, some scheduled events can become *irrelevant* due to the system's evolution and have thus to be deleted.

The practical implementation of this list can be done in various forms. The two most widely used implementations are:

– (doubly) *chained list*: it is a linear structure. The average insertion time (i.e. to find the good place to insert a new event) is of $O(N)$ where N is the total number of the events in the list;

– *binary tree*: it is generally implemented as a *heap*. This structure offers an average insertion time of $O(\log_2(N))$ where N is the total number of the events in the list.

There is also a structure known as a *calendar*. Roughly speaking, this method consists of cutting time axis into a number of segments which are renewed as the Time advances. Each segment is divided into a number of sections. We can compare this method with a *calendar* for which the segment is one year and the sections are days of a year. Thus, just as we can note an appointment directly on the concerned day's page, we can directly record an event on the good section. The advantage of this process is the speed of reaching the good time section. It is however necessary to choose the size of the segment well and the granularity of the sections.

In terms of organization, the handling of the events can be done:

– in a centralized way: there is a single list for all of the events generated by all of the entities;

– or in a distributed way: each entity has its own list of the events generated by it.

In all the cases, the events are carried out according to the chronological order (in the sense of simulated time). If several events take place at the same instant, as the scheduler works in an ordered way, these events will be invoked sequentially and the order of invocation may not coincide with their insertion order. This should not be a problem, since the real world works in the same way. The single important point is that all these events take place at the same (simulated) time. Nevertheless, certain programming errors come from a bad comprehension of this nature.

The implementation of a scheduler is a rather basic programming task, which can be done by average programmers through various programming language. This undoubtedly constitutes one of the reasons of the diversity and proliferation of the simulation languages.

The scheduler constitutes the core of any language of simulation. It is in general an internal function that an ordinary user does not have to directly deal with. However, since it constitutes the foundation of the DES, it is interesting to know how it works.

4.3. Generators of random variables

Any simulation language should provide at least one PRNG of $U(0,1)$ of which it is generally possible to specify the seed (at least at the launching of simulation).

It is then possible to generate sequences of r.v. of various distributions. Most simulation languages provide generators for some commonly used distributions (exponential distribution, geometric distribution, etc.).

In certain languages, it is possible to set up a different generator for each r.v. Then each generator corresponds to a sub-sequences of the internal generator, which makes it possible to guarantee independence, during

simulation, between the r.v. which are in the reality independent quantities (see section 3.4.5.3, where an example with OMNeT++ is also presented).

4.4. Data collection and statistics

Certain languages (for example OPNET or OMNeT++) provide the functionality of data collection for variables of our choice. Some languages even provide the coupling of these statistical observations with the control of simulation (stop of the simulation when some target statistical precision is reached).

Other languages (for example NS2) do not provide any statistical tool. For these one, we must ask the simulation language to record data in a file (trace) during the simulation for an appropriate post-simulation statistical analysis.

It is always important to be able to record raw data, so that they can be submitted to more in-depth and specialized statistical analysis. There are specialized tools (for example the R language) to carry out various statistical processing.

4.5. Object-oriented programming

The capacity of providing a detailed modeling of the systems, through its components and mechanisms, makes the force of simulation compared to the analytical method.

A simulation language must be provided with a high capacity of algorithmic and structural description. That imposes practically the use of an OOP language. In practice, there are the choice between:

– a specific language: for example the Simula language which is the ancestor of the object-oriented languages;

– or a general-purpose language, for example C++, to which we add functionalities (for example scheduler) and specific components (for example queue, customer, etc.). This is the case of OMNeT++.

The OOP language makes it possible to build objects with an algorithmic implementation. Some objects are basic blocks shared by all, such as queues,

customers, etc. A number of simulation languages are designed for a particular application domain, they come with pre-built models specific to this domain.

4.6. Description language and control language

Once we achieve the programming of a model, the latter has to be configured with various parameters (corresponding to various real-world situations). In addition, simulation runs must be controlled (for example, the stop time, choice of the seed, etc.).

In order to facilitate these experimental operations, the majority of the simulation tools work with two languages:

– a description language, such as we presented in the preceding section. It generally concerns a compilation language. In `OMNeT++`, it is `C++`;

– a language for control and configuration, which is generally an interpreted language. In `OMNeT++` for example, it is a specific language `NED` which is used for the macroscopic description and construction of the models.

4.7. Validation

4.7.1. *Generality*

A crucial question is to ensure the correctness of a simulation program. First of all, it has to be noticed that a simulation model is inherently a computer program. So there are, by definition, “bugs”, i.e. syntactic or semantic errors. We have to proceed to the usual “debugging”. However, simulation programs have their specificities that we will present in this section.

Basically, validation of a traditional computer program often consists of checking if this one performs the required functionalities well through a list of predefined outcomes. However, a simulation model aims precisely to study the unknown behavior of a system. Thus, we can not use the traditional method, as there is no precise reference to validate a model. This constitutes one of the principal difficulties for the validation of a simulation model.

4.7.2. *Verification of predictions*

Nevertheless, there are particular situations where the outcomes of a simulation can be predicted. These situations facilitate the detection of bugs and/or design errors. Here are the principal ones:

– checking of the *invariants*. They are numerous, such as:

- flow conservation;

- sum of the probabilities;

- Little's law;

– case for which *theoretical result* exist (e.g. $M/M/1$ or $M/G/1$ queues);

– *extremes case* (reduced topology, impulsional traffic with a single packet, overload, etc.) whose characteristics allow us to predict the result by qualitative analysis;

– *expected results* from the real world. If we have results for some scenarios by real measurements, we can also run simulations for these ones, in order to check whether the simulation model reproduces the awaited results as it should do;

– *inspection* of data produced by simulation:

- display the results *graphically*: this facilitates, in a straightforward way, the detection of situations which are *a priori* abnormal (discontinuities, etc.); if the simulation language itself does not provide such functionality, it is always possible to use an external tool,

- *trajectory analysis*: the data memorized in a trace file allow us to follow the trajectory of a given customer and the related events. This work could be *rather tedious*, but it really helps to identify possible design errors and/or programming bugs,

- in a similar way, but with less effort (and also less insight), if possible, is to follow graphically the trajectory of a customer. In `OMNeT++` for example, there is the possibility to display graphically all the events (see Figure 2.1).

As a synthesis, we can note that these approaches have in common the *a priori* knowledge of some awaited results, against which we compare the data produced by the simulation program.

4.7.3. *Some specific and typical errors*

One of the traditional sources of errors is the use of (simulation level) global variables. It is basically a bad coding practice. In the case of simulation, the use of global variables often lead to semantic errors. This is particularly true for the case of computer networks. Indeed, we are basically in a distributed system, inherently, there is *no* such physical quantity which would be shared. Hardly could we consider the (simulated) Time (the wall clock) as a shared quantity. It is considered as a global variable by necessity due to the need for events handling. But, it should be known that, in the real world, local times (your watch and that of your neighbour, for instance) are not exactly the same and the so-called "wall clock" (or global time) can be obtained only through synchronization mechanism.

A second source of errors is the negligence of the importance of the events which make time advance, even with small steps. For example, in the *Token Ring* protocol, a right to transmit (the token) circulates among the entities, only the holder of the token can send messages. The passing of token from one entity to its downstream neighbor takes a certain amount of time which is quantitatively negligible compared to the sending of a message. However, we cannot neglect this time in simulation because each token-passing makes the (global) time advance. If, at some instant, every entity is waiting for a future message, to neglect this token-passing time would *freeze* the time, and so keep the system in its *status quo* for ever. This would cause a fatal error. In section 6.10, we present a model of the Token Ring.

A third source of errors is due to the *parameter setting*, even though it is not really a *coding* problem, but rather a problem related to the simulation configuration. Actually, the set of parameters is composed of (1) the designer's own choice and (2) the parameters of existing modules that are *re-used* in the building of the current model. So, it is also a matter of model/program architecture. Two main types of errors are:

– *inconsistency* between the *default* parameter setting of some re-used modules and the correct parameters for *your* model: actually, existing modules come with their parameters. For many modules (e.g. routing component in the Internet, switch, TCP protocol entity), the number of parameters can instead be significant, just as their counterpart in the real world. Most of them are set with some *default* values. So, when you re-use a model, not only do you inherit the algorithmic behavior, but also a *particular* case of this behavior by *default*. Of

course, we can always modify the setting of parameters. The two difficulties are:

- this requires a very careful checking of the parameters of the re-used module and a good mastering of the underlying real counterpart;

- additionally, not all of these parameter settings are well documented and/or easily accessible: it depends on the coding practice of the module designer.

For this reason, when the goal is to study a punctual and isolated aspect of a big system, there is a real trade-off, in terms of *modeling effort*, between taking an existing module of the system which is *comprehensive* but rather complex and developing a *customized* simple model focusing on the punctual problem.

– *handling* of various scenarios: this is much more of a problem related to simulation running. As the set of parameters can be rather large, and each run of simulation can be characterized by a consequent number of particular parameters, the risk would be too big if we set parameters "by hand" in an *ad hoc* manner. It is recommended to write an auxiliary scenario-generator, in order to get a *systematic* parameter setting.

4.7.4. *Various tests*

4.7.4.1. *Parameter setting and continuity*

A good practice consists of running simulation on a whole range of parameters. We previously mentioned the extreme cases. It is also useful to test various intermediate values to observe a possible lack of continuity, which could result from an error of modeling or programming.

4.7.4.2. *Robustness*

Another good practice consists of replacing the model of a subentity by another one, in order to test the robustness of the whole model. This is particularly useful if the model used for the subentity is subjected to caution. That can be due to the difficulties of getting a satisfactory model. That can also be due to the voluntary choice of a simplifying model. In this case, simulate the system with another model makes it possible to see the impact of the choice of the model. For example, for the distribution of the packets' size, we can successively test the assumption of a Pareto distribution followed by that of a log-normal.

5

Simulation Running and Data Analysis

In this chapter, we emphasize the fact that each simulation run is a statistical experiment. This requires a certain number of precautions for the control of the experiment (simulation run), as well as the analysis of the simulation outcomes. It is necessary to consider the statistical quality of the collected data. We present some basic techniques allowing us to obtain simulation results with a quantifiable and acceptable quality through a simulation run of adequate duration.

5.1. Introduction

First of all, we recall that each simulation run is a particular statistical experiment carried out through a computer program which is a virtual version of the target system. In order to undertake a simulation run well, we must pay attention to:

– the configuration of the system so that the system works under the correct regime (generally in stationary regime) and with the right set of parameters: their number can be important, it is recommended to proceed with a careful and systematic check in order to avoid useless runs, or, worse, a misleading wrong result from a wrong configuration that we take as the target one;

– the set of data to be collected and the collection methods: this concerns trade-off between the ability of focusing on some details of the system and the efficiency of the simulation. The former leads to a preselection of a large set of potential interesting parameters during the programming of the model, whereas the latter recommends just selecting a right subset of them. Some languages (for instance `OMNeT++`) allow the actual selection, for each run, of a particular subset of quantities to be collected, assuming that, for each potential candidate-quantity, the collection mechanism has been programmed;

Performance Evaluation by Simulation and Analysis with Applications to Computer Networks, First Edition. Ken Chen. © ISTE Ltd 2015. Published by ISTE Ltd and John Wiley & Sons, Inc.

– when to *stop the simulation*: this is a fundamental question for each simulation run. The ending condition should be carefully set, since it concerns both the simulation efficiency and the quality of the simulation output. We will give a detailed discussion on this issue;

– what is the *statistical quality* of the output: the latter is usually assessed through the *confidence interval*. A good statistical study often requires multiple independent experiments. Each simulation run provides a particular observation of the system. In order to obtain an acceptable statistical quality, it is often necessary to carry out multiple runs with long simulation duration.

5.2. Outputs of a simulation

5.2.1. *Nature of the data produced by a simulation*

Once launched, a simulation program continuously produces data related to selected indicators through which the system will be studied. Let us take the example of the routing of Internet Protocol (IP) packets through the Internet. We can be interested in:

– the sojourn time of each packet in a given router: this one is provided directly by the target router;

– the end-to-end delay (between a source and a given destination): for it to be obtained, it is necessary to memorize the sending time of the packet IP at the source (a usual way is to append this time as auxiliary, and artificial, information to the packet, as allowed by the object-oriented programming), the end-to-end delay time can thus be deduced by the destination upon arrival of the packet;

– the rejection ratio in a given router: this ratio is measured *via* an auxiliary indicator which records, for each packet, its binary status (routed or rejected).

From the mathematical point of view, these physical quantities constitute each one a stochastic process (see section 14.1), denoted as $X(t)$, i.e. a sequence of r.v. indexed by time. Given a simulation running on the simulated time interval $[0, T]$ under some initial conditions denoted by s. For each observed quantity X, each simulation run under initial conditions s produces a sequence of data $\mathcal{S}(s, T) = \{X_s(t_i)\}_{i \geq 1, t_i < T}$ where $X_s(t_i)$ is the value produced at the simulated moment t_i of this run. The objective of a simulation is to get some statistical conclusions on X from L sequences $\{\mathcal{S}(s_i, T)\}_{i=1,\ldots,L}$, s_i being the initial condition of the i-th run out of a total

of L simulation runs. In the remaining of this chapter, in order not to overload the notations, we omit the indication of the initial conditions s except otherwise indicated.

5.2.2. *Stationarity*

A fundamental question in the study of a stochastic process is to know whether this process is stationary, i.e. if a conclusion for a given moment is still valid at other moments. The majority of the studies are done under stationary regime (also as known *steady state*).

To be completely rigorous, X constitutes a stationary process if it verifies:

$$\forall t, \forall h, \forall x, \quad F(x,t) = F(x,t+h) \qquad [5.1]$$

where $F(x,t)$ gives the distribution of X at x and instant t.

This criterion is not easily verifiable in practice, because it is difficult (even impossible) to check formula [5.1] for every moment t, over all the durations H, and on all points x.

If a process is stationary, this must be the case for its first moments also (the reciprocal is, of course, false). So, in practice, we have to deal with *weak-sense stationary*, i.e. the asymptotic stationarity of the first two moments of a process:

– we have the asymptotic stationarity of the first order if:

$$\exists\, C, \forall \epsilon > 0, \exists\, t_0, \text{such that } \forall t > t_0, \left| E[X(t)] - C \right| < \epsilon$$

In other words, the mathematical expectation (mean value) converges toward a constant C: $\lim_{t\to\infty} E[X(t)] = C$.

– A more elaborate control consists of calculating the autocorrelation between $X(t)$ and $X(t+h)$. We have the asymptotic stationarity of the second order if:

$$\exists\, f(.), \forall \epsilon > 0, \exists\, t_0, \text{such that } \forall t > t_0, \forall h, \left| \mathrm{Cov}[X(t), X(t+h)] - f(h) \right| < \epsilon$$

In other words, $\lim_{t\to\infty} \mathrm{Cov}[X(t), X(t+h)] = f(h)$. In particular, for $h = 0$, we find the variance which converges toward a constant.

In practice, even these properties cannot be completely checked. Indeed, $E[X(t)]$ should be checked against all the possible scenarios, i.e. all the initial conditions, or, at least, a large number of them. This would require a tremendous number of simulation runs.

We are generally reduced to checking the expectation stationarity of the sequence $\mathcal{S}(s,T) = \{X_s(t_i)\}_{i\geq 1, t_i<T}$, denoted as $\overline{X}(s,T)$. As $\mathcal{S}(s,T)$ can be considered as a sample, the expectation is calculated by the sample mean formula. In order to ensure the stationarity of $\overline{X}(s,T)$ with a given precision ϵ, we have to find a threshold instant t_0 such that:

$$\left|\max\{\overline{X}(s,T_i)/T_i \in [t_0,H]\} - \min\{\overline{X}(s,T_i)/T_i \in [t_0,H]\}\right| < \epsilon \quad [5.2]$$

where H is the ending instant of the simulation. We can then *reasonably assume* that the process is in stationary mode from t_0. The initial interval $(0,t_0)$ is called the *transient period* of the process, i.e. the time interval before the process starts to converge toward its stationary regime. If we are able to check equation [5.2] with the precision ϵ, we will be able to suppose that the stationary regime is reached with the precision ϵ.

When the system is stationary, it is theoretically possible to run only one simulation which would be sufficiently representative of the behavior of the system, if the simulation runs for a sufficiently long duration. In practice, however, it is necessary to run several replications.

For each simulation campaign (generally with several runs with different seeds), two fundamental questions arise:

– How do we fix H, the end of the simulation? We will see that the end of a simulation is determined primarily by the statistical quality of the results which will have to be in adequacy with that required by the objective of study.

– How do we locate the transient period (t_0) and how to deal with the data of this period? We will see that this is done usually in a rather *ad hoc* manner.

5.2.3. *Example*

We present below a set of curves as an example. The studied system is a simple $M/M/1$ queue (see section 9.2) with $\lambda = 1/6\,s$ and $\mu = 1/3\,s$,

i.e. $\rho = \frac{\lambda}{\mu} = 0.5$. We measure $E[N]$ the mean size of the queue. Theoretically, $E[N] = \rho/(1-\rho) = 1$. Simulations have been run for a duration of $T = 1000$ s in simulated time. In the figures, the vertical lines give the instantaneous values of $N(t)$ whereas the curve gives the sample mean of N up to t.

Figure 5.1 presents the usual case of an initially empty queue. We can locate t_0, the end of transient period, at around $t_0 = 600$ s.

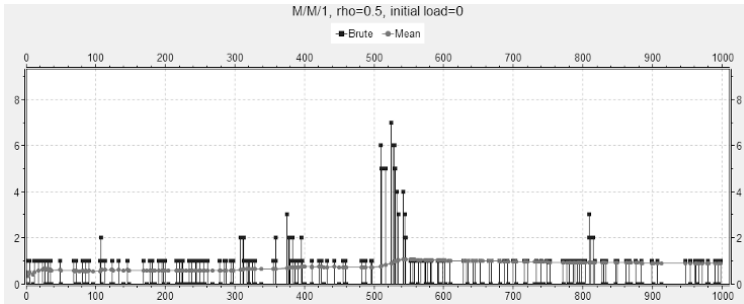

Figure 5.1. *Initial customer number:* 0

Figure 5.2 presents the case where the queue initially has 10 customers. We can notice that the transient period becomes much longer. At the end of simulation, the sample mean is always in a decreasing phase toward 1 without stabilization. Simulation should be conducted longer than $T = 1000$ s.

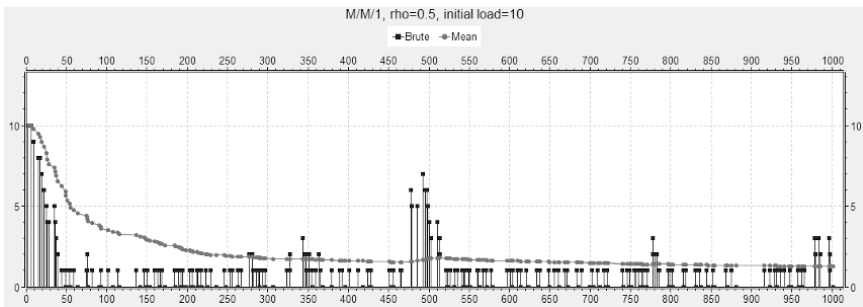

Figure 5.2. *Initial customer number: 10*

We will see the effect of the duration of simulation through the following two situations ($T = 1000$ s and $T = 2000$ s) for an initial load of five customers. We can see that, under this particular situation and for this

particular run, there are significant "fluctuations" around $t = 500$. With $T = 1000$ (see Figure 5.3), the curve is still not stabilized around 1 at the end of the simulation; with $T = 2000$ s (see Figure 5.4), starting from $t_0 = 1200$ s, we can observe a convergence which is maintained.

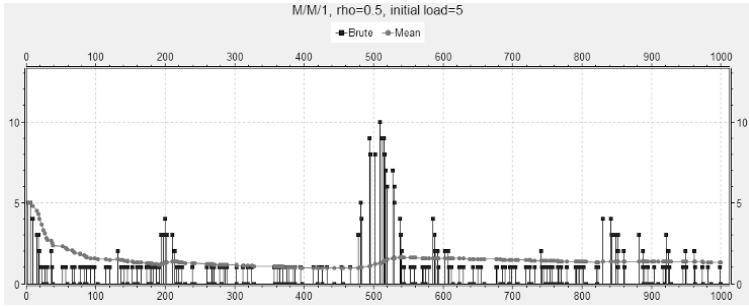

Figure 5.3. *Initial customer number: five,* $T = 1000$ *s*

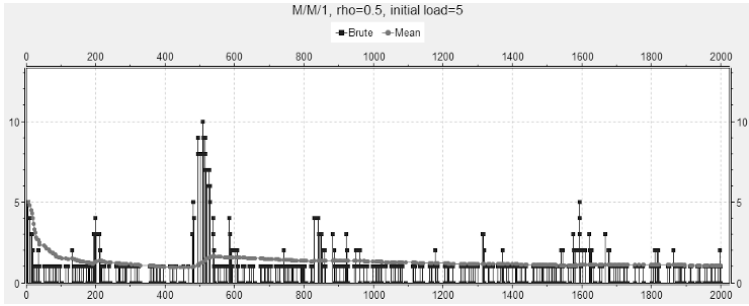

Figure 5.4. *Initial customer number: five,* $T = 2000$ *s*

5.2.4. *Transient period*

Most of the time, we are interested in the behavior of a stationary mode. Thus, it is necessary, as much as possible, to eliminate the effect of the transient period which depends largely on the initial conditions. This is generally done by discarding data of this period (which corresponds to the beginning of simulation), in order to minimize the impact of the initial conditions.

The problem is that there is no formula which can decide where the transient period ends. Thus, this is done in an *ad hoc* way, under the guideline of the operational formula [5.2]:

– *graphic method*: it is an intuitive method which consists of locating graphically the transient period, by drawing the sample mean of the target quantity. The stationary regime is considered to be reached if the curve becomes stable with weak variations around a central value C i.e. the curve is almost flat (horizontal) around C;

– *automated detection*: the criterion given by formula [5.2] can be automatically checked, by regularly comparing the mean values of adjacent intervals, $T_1, T_2, \ldots, T_i$. If, starting from a certain i, we note a weak variation of the temporal averages, we can try to obtain a confirmation of the tendency of the following measures. If the tendency is confirmed, we can consider that the stationary regime is reached;

– *by anticipation*: the first approach is easily carried out but it is more suited to an *a posteriori* control. The second approach needs some specific function which is not provided by all the simulation languages. A common practice consists of fixing, in an *a priori* manner, the transient period. A generally used *ad hoc* rule consists of discarding data belonging to the initial period $[0, t_0]$ in a proportion which should not exceed 10% of the total simulation duration (denoted T), i.e. $t_0 \leq 0.1T$. Certain simulation languages provide this facility as a basic function (see section 6.9.3 for an example in `OMNeT++`). If this function is not available, this method has to be coded directly, and individually for each quantity, in the simulation program.

In all the cases, it would be necessary to check, *a posteriori*, the relevance of the choice of t_0. Moreover, we are never certain to be able to completely eliminate the impact of the transient period. This explains the need for carrying out simulations over long durations (in terms of simulated time), and/or under various initial conditions (in particular starting from a state close to that of the stationary state), in order to minimize the influence of the transient period.

5.2.5. *Duration of a simulation*

Let us recall once again that a simulation is primarily an experiment which provides a particular sample of observations from where we deduce a statistical conclusion. Therefore, the duration of such an experiment is mainly dictated by

the quality of the results, i.e. in practice, the confidence interval of the results. The duration of a simulation thus cannot be decided in advance but must be readjusted according to the data provided by the simulation. In particular, we should not pay attention to simulated time but rather to the number of events, i.e. the quantity of the statistically significant data produced by simulation, in order to reach a certain statistical quality (indicated through the confidence interval).

In addition to this principal concern of statistical quality, it is also important to minimize the duration for reasons of efficiency. There is a true trade-off between the quality and the duration of a simulation. It is indeed obvious that the longer the duration is, the more exhaustive the statistical study will be and its results refined. At the same time, a long simulation is surely more expensive, not only in terms of resource consumption but also in terms of human involvement.

This trade-off can be achieved by seeking a *reasonable* degree of accuracy on the statistics. After all, it is about a modeling and not about the system itself. A modeling process often brings us to adopt simplifications or approximations. Thus, it is generally not so useful to claim a very high degree of accuracy. For each observed quantity, it is also necessary to examine the objective of study to determine the useful confidence interval. Consider, for example, the case of a study whose goal is to know if a mean delay remains below 200 ms, if the result is 170 ms with a confidence interval at ± 10 ms, it should be sufficient. On the other hand, if the confidence interval is ± 30 ms, then additional runs must be done for a better precision.

5.3. Mean value estimation

In a simulation, we deal mainly with the estimation of the mean values of various quantities. Let us take again the example of a router where we are interested in the mean number of packets in a buffer. We may be interested in a ratio (or proportion), for example, the rejection ratio of the packets due to overload, or the proportion of time when the buffer is not empty. We will see that all these calculations are by nature a matter of *mean value estimation*. The latter thus occupies a central place in the analysis of simulation data (see section 6.9.1.5 for concrete examples under `OMNeT++`).

5.3.1. *Mean value of discrete variables*

First, we will deal with the cases of the discrete variables. Let us take a quantity X, for example, the number of packets in a router. Simulation produces a sequence $\{x_i\}_{i=1,...N}$ N values, x_i being the i-th outcome of X.

The problem is then to estimate, from $\{x_i\}_{i=1,...N}$, the mean value $E[X]$, as well as the variance $Var(X)$ of the underlying distribution. The theory of statistics (see Chapter 13) provides estimators for sample mean $\overline{X}$ (see equation [13.3]) and sample variance S^2 (see equation [13.4]) which we present here:

$$\begin{cases} \overline{X} = \frac{\sum_{i=1}^{N} x_i}{N} \\ S^2 = \frac{\sum_{i=1}^{N}(x_i - \overline{X})^2}{(N-1)} = \frac{\sum_{i=1}^{N} x_i^2 - N\,(\overline{X})^2}{(N-1)} \end{cases}$$

The above formulas suggest that it is not necessary to memorize all the sequence $\{x_i\}_{i=1,...N}$ during a simulation. We only need to memorize some key-values. To implement the estimators $\overline{X}$ and S^2, it suffices to update, progressively during the gathering of measurements x_i, the current values of:

– N, which is obtained by incrementing a counter `N` (`N++`);

– $A = \sum_{i=1}^{N} x_i$ with `A:=A+x_c` where `x_c` is the current value of x_i ;

– $B = \sum_{i=1}^{N} x_i^2$ with `B:=B+(x_c)*(x_c)`.

We can then provide, at any time, $\overline{X} = A/N$ and $S^2 = \frac{B - A^2/N}{N-1}$. This method has the advantage of allowing the first two moments being estimated in a very compact way and has the disadvantage of losing all trace ($\{x_i\}_{i=1,...N}$) of the underlying process.

We would then think that it were possible to calculate the confidence interval at $1-\alpha$ of X (see section 13.3.6) with:

$$\left[\overline{X} - z(N-1,\ \alpha/2)\sqrt{S^2/N},\ \ \overline{X} + z(N-1,\alpha/2)\sqrt{S^2/N}\right]$$

by assimilating $Var(X)$ to its estimator S^2. However, this is *wrong*, and the approximation of $Var(X)$ by its estimator S^2 is far from being the only source of error. Indeed, the formula of the confidence interval (resulting from the

central limit theorem, see section 12.8.3) could be applied *only if* $\{x_i\}_{i=1,...N}$ are i.i.d. r.v. In our situation, the assumption of an identical distribution can be accepted since $\{x_i\}_{i=1,...N}$ actually are outcomes of the same system, on the contrary, the assumption of independence cannot be verified in general. In a router, for example, the size of a packet has an impact on its transmission time and thus on the number of packets arriving during its transmission. In most of the cases, we cannot obtain i.i.d. r.v., so the confidence interval cannot be calculated like this. We will address this topic in section 5.3.4.

5.3.2. *Mean value of continuous variables*

When the measured quantity is a continuous variable, for example, buffer occupation $X(t)$, the brute data are $\{X(u), t_0 \leq u \leq t_0 + D\}$ where t_0 is the end of the (assumed) transient period (i.e. beginning of the data collection), and D gives the duration of the period. The following formula gives the temporal mean of $X(t)$ at D:

$$\overline{X(D)} = \frac{1}{D} \int_{t_0}^{t_0+D} X(u)du \qquad [5.3]$$

It should be noted that in a simulation with discrete events, $X(u)$ appears rather in the form of a step function whose value changes on a finite number of points.

5.3.3. *Estimation of a proportion*

If we want to calculate the packet rejection ratio in a router, we can simply do it through the r.v. X defined as follows: $X = 1$ if the packet is rejected, if not $X = 0$. The mean value of X gives precisely the rejection ratio. For example, if there are 10 rejected packets over a total of 100, the mean value gives 0.1 which corresponds to the proportion (ratio) of the rejected packets. We see that the calculation of a ratio can be assimilated to that of the mean value of a binary r.v.

In the same manner, if we want to calculate the proportion of time when the buffer is not empty, just calculate the temporal mean of $X(t)$, where $X(t)$ is an indicator function such as $X(t) = 1$ if the buffer is not empty, $X(t) = 0$

otherwise. Once again, the calculation of a proportion can be assimilated to that of the mean value.

5.3.4. *Confidence interval*

We have just seen the methods allowing the computation of sample mean $\overline{X}$ of a quantity X from a set of data $\{x_i\}_{i=1,...N}$ provided by a simulation.

The problem consists of providing an estimate of the true value $E[X]$, with the indication of a degree of accuracy on the obtained estimate. In statistics, it is a question of obtaining a confidence interval.

Each simulation run, say the i-th, provides an estimate, say M_i, of $E[X]$. Each M_i behaves as a random variable. It is then a question of estimating the variance between these observations M_i, i.e. to estimate the variance of the estimator.

The available theoretical results about confidence interval estimation assume independence between the estimated values. Thus, we must pay attention to ensure the independence between these estimates M_i. The following section presents some commonly used methods: the replication method, batch-means method and regenerative method. They have in common this concern of ensuring, in one way or another, independence between the estimates.

5.4. Running simulations

5.4.1. *Replication method*

The principle of this method consists of launching several independent simulations with the same simulation configuration (but with different seeds). Thus, it concerns the same system. Next, we consider such a campaign with R simulations, which thus produce R samples of the same stochastic process.

Given X the r.v. of which we want to estimate $E[X]$. Denote $\{x_{k,i}\}_{i=1,...,N}$ the sequence of the k-th simulation, $1 \leq k \leq R$ (idem in the remainder of this section). The R sequences are obtained in the same way, in particular, they have the same transient period.

We can calculate, for each simulation k, an estimate of X related to this simulation, $\overline{X_k}$, with the classical sample mean formula:

$$\overline{X_k} = \frac{\sum_{i=1}^{N} x_{k,i}}{N}$$

The $\{\overline{X_k}\}_{k=1,\ldots,R}$ constitute i.i.d. r.v. Indeed, they relate to the same system (thus, the same distribution) and are independent by construction[1] (obtained from different simulations). We calculate the sample mean of $\{\overline{X_k}\}_{k=1,\ldots,R}$, denoted by M, with:

$$M = \frac{\sum_{k=1}^{R} \overline{X_k}}{R} = \frac{\sum_{k=1}^{R} \sum_{i=1}^{N} x_{k,i}}{NR}$$

which is also an estimator of $E[X]$ and we calculate the variance of $\{\overline{X_k}\}_{k=1,\ldots,R}$ as follows, denoted by S^2:

$$S^2 = \frac{1}{R-1} \sum_{k=1}^{R} (\overline{X_k} - M)^2 = \frac{1}{R-1} [\sum_{k=1}^{R} \overline{X_k}^2 - R\,M^2]$$

As $\{\overline{X_k}\}_{k=1,\ldots,R}$ are i.i.d. r.v., the formula of the confidence interval can be applied:

$$E[X] \in \left[M - z(R-1, \alpha/2) \frac{S}{\sqrt{R}},\ \ M + z(R-1, \alpha/2) \frac{S}{\sqrt{R}} \right]$$

i.e. by considering $M = E[X]$, the error margin is lower than $z(R-1, \alpha/2) \frac{S}{\sqrt{R}}$ with a confidence degree of $1 - \alpha$.

The drawback of this process is that it is necessary to carry out a significant number of replications to approach the condition of applicability of the formula (convergence). This constitutes in practice a serious difficulty.

1 This property is still subject to discussion. Some people think that we cannot consider them as completely independent.

5.4.2. *Batch-means method*

The principle of this method (see [FIS 01], p. 247) consists of cutting out data resulting from the same simulation in blocks of equal size. Let us suppose that a simulation produces L outcomes, the method can be described as follows:

– cut out the L outcomes in B blocks of N outcomes each ($L = NB$);

– for each block k, ($1 \leq k \leq B$, idem in the continuation of this section), calculate the sample mean of the block M_k over outcomes $\{x_i\}_{i=N(k-1)+1...Nk}$ with:

$$M_k = \frac{\sum_{i=N(k-1)+1}^{Nk} x_i}{N}$$

– calculate the sample mean (M) and the sample variance (S^2) on the M_i with:

$$\begin{cases} M = \frac{\sum_{k=1}^{B} M_k}{B} = \frac{\sum_{i=1}^{NB} x_i}{NB} \\ S^2 = \frac{\sum_{k=1}^{B} (M_k - M)^2}{B-1} \end{cases}$$

Until this stage, we followed the same process as for the replication method, here the blocks replace various simulations. The M_i is identically distributed for the same reasons as previously. If, moreover, the blocks are independent, the central limit theorem applies to it, the confidence interval at $1 - \alpha$ is:

$$\left[M - z(N-1, \alpha/2)\sqrt{S^2/N},\ \ M + z(N-1, \alpha/2)\sqrt{S^2/N}\right]$$

However, there is no *a priori* reason to think that the blocks are independent, because the blocks are contiguous and are within the same simulation. The choice of the blocks was the subject of many studies and several methods were developed in order to ensure their independence. We present one of them below (see [BAN 05], p. 420). This method consists of forming, at the first stage, about $B = 100$ blocks, then to calculate the autocorrelation:

$$\hat{\rho_1} = \frac{\sum_{i=1}^{B-1} (M_i - M)(M_{i+1} - M)}{\sum_{i=1}^{B} (M_i - M)^2}$$

– if $\hat{\rho_1} \leq 0.2$, then the hypothesis of independence of the blocks is accepted. We carry out an effective calculation with a reduced number of blocks (typically, $B \in [30, 40]$ blocks), by an expansion of the initially formed blocks;

– if $\hat{\rho_1} \geq 0.2$, it is necessary to increase the duration of simulation (in a proportion from 50% to 100%) and then we restart the same method.

5.4.3. *Regenerative method*

The main issue with the batch-means method is that the blocks are not *a priori* independent. Thus, there is always a doubt on the confidence interval. The *regenerative method* aims at obtaining, by construction, the independence of the blocks.

Without entering into the mathematical details, let us say that independence is guaranteed if cutting is done at moments called *regeneration point*. These are the boundaries which eliminate any impact of a previous block to the incoming block. Thus, they are the points which create a rupture between the past and the future. Each block constitutes a *regeneration cycle*; a good candidate for regeneration points is the instant where a system becomes active, i.e. it leaves the idle state.

We briefly describe in the next the various stages of calculation (see [JAI 91], p. 433). The notations and assumptions used are: there is a total of N cycles; the number of outcomes of cycle i is denoted by n_i; outcome j of cycle i is denoted by x_{ij}, $j = 1, \ldots, n_i$. Here is the method:

1) calculate the sum on cycle i, denoted as S_i, with:

$$S_i = \sum_{j=1}^{n_i} x_{ij}$$

2) calculate the sample mean (M) over all of the outcomes with:

$$M = \frac{\sum_{i=1}^{N} \sum_{j=1}^{n_i} x_{ij}}{\sum_{i=1}^{N} n_i} = \frac{\sum_{i=1}^{N} S_i}{\sum_{i=1}^{N} n_i}$$

3) calculate the cumulative difference between each of the n_i outcomes x_{ij} and M for cycle i, denoted as D_i, with:

$$D_i = S_i - n_i M$$

We can verify that the sample mean of D_i is zero;

4) calculate the sample variance of the D_i, denoted as S_D^2, with:

$$S_D^2 = \frac{\sum_{i=1}^{N} D_i^2}{N-1}$$

5) calculate the mean number of outcomes per cycle, denoted as L,

$$L = \frac{\sum_{i=1}^{N} n_i}{N}$$

6) the confidence interval at 95% of the mean value is given by:

$$(M - 1.96\frac{S_D}{L\sqrt{N}}, M + 1.96\frac{S_D}{L\sqrt{N}})$$

The main concern with this method is the definition of the regeneration points, and, more importantly, how to effectively implement the method.

5.5. Variance reduction

We have just seen that the quality of the estimate is measured through the width of the confidence interval. The latter depends on the variance and the number of observations N. However, the reduction factor of the latter is $1/\sqrt{N}$, it would be very expensive to get a tighter confidence interval only by increasing N. Techniques of *variance reduction* were developed for this purpose.

We insist here on the fact that these techniques are sophisticated statistical methods which are very delicate to handle and implement. Poorly applied, this kind of technique could even produce the opposite effect.

5.5.1. *Common random numbers*

We choose to present this technique first because it is rather easy to implement. It is to be pointed out that it is mainly used in the comparison of two similar systems.

The following scenario illustrates the principle of this technique. Let us imagine that we seek to compare the average sojourn time (D) of two queues A and B, A is ordered as First-In-First-Out (FIFO) and B as Last-In-First-Out (LIFO).

The *common random numbers* (CRNs) method consists of sending the same arrival sequence to the two queues, with the same service time. In fact, it is the bifurcation of the same sequence to the two queues. Let us suppose that we run N simulations and the i-th simulation of A (respectively, B) produces an estimate of $E[D]$, denoted by $\overline{D}_{i,A}$ (respectively, $\overline{D}_{i,B}$). At the end of these N replications, the sample mean and the sample variance of D in A are given by the classical formulas:

$$\overline{D}_A = \frac{\sum_{i=1}^{N} \overline{D}_{i,A}}{N}, \quad \overline{S}_A^2 = \frac{\sum_{i=1}^{N} (\overline{D}_{i,A} - \overline{D}_A)^2}{N-1},$$

We do not give the explicit formulas of B which are identical except for the index B.

The comparison between A and B consists of calculating the difference $\Delta = \overline{D}_A - \overline{D}_B$ which is an estimator of $E[D_A] - E[D_B]$. As all the estimates are in fact r.v. Δ is also an r.v. The CRN method allows us to reduces its variance. Indeed:

$$Var[\Delta] = Var[\overline{D}_A] + Var[\overline{D}_B] - 2\,\mathrm{Cov}[\overline{D}_A, \overline{D}_B]$$

If we had run two simulations separately, i.e. in particular with different arrival instants and service times (which obey nevertheless the same arrival and service distributions), the results would have been independent and thus $COV[\overline{D}_A, \overline{D}_B] = 0$.

With CRN, the two simulations are correlated. We will see that this correlation is in fact positive. Indeed, an increase in traffic will lead to an increase in the number of customers both for A and B, and thus causes an

increase in $E[D_A]$ and $E[D_B]$. Thus, $Var[\Delta]$ is lower in the case of the simulations done with a common sequence than that in the case of independent simulations. The true difference between $E[D_A]$ and $E[D_B]$ is then better assessed due to this technique.

We give some remarks to point out the delicacy of the application of this method. If the service time was not the same for A and B, i.e. it had not been generated at the source but had been generated separately by A and B, we would not really be in a situation of the common sequence. If we wanted to compare the two queues, of which one has a limited capacity and thus with possible rejection of certain customers, we would not have the same sequence either. This shows the limits of this kind of technique which would be even more restrictive in the following method.

5.5.2. *Antithetic variates*

The principle of this method is simple and easy to understand. We want to estimate $E[X]$ of some r.v. denoted as X. Given $\overline{X_1}$ an estimate of $E[X]$. $\overline{X_1}$ results from a sequence of observations which are themselves a sequence of events generated from a sequence of basic r.v. following $U(0,1)$, say $\{u_i\}_{i=1...L}$.

Let us run the same simulation again, but with an r.v. sequence which is complementary, obtained from a sequence of complementary basic random variables $\{v_i = 1 - u_i\}_{i=1...L}$. If the generation of the r.v. is done by the inverse transformation method (see section 3.5.1), it is observed that the sequence $\{v_i = 1 - u_i\}_{i=1...L}$ will produce values locating in the side opposed to those generated in the sequence $\{u_i\}_{i=1...L}$. As an example, let us take an exponentially distributed interarrival time with $\lambda = 1$. If in the first simulation the value was $-\log 0.1 = 2.30$, its corresponding value in the second simulation would be then $-\log 0.9 = 0.105$. Two simulations should thus proceed under opposite situations thus *negatively* correlated.

Let $\overline{X_2}$ be the estimate of $E[X]$ at the end of this second simulation. We have two r.v., $\overline{X_1}$ and $\overline{X_2}$. We define $\Sigma = 1/2(\overline{X_1} + \overline{X_2})$. It is clear that

$E[\Sigma] = E[X]$. The variance of E is given by:

$$\begin{aligned} Var(\Sigma) &= 1/4\Big(Var[\overline{X_1}] + Var[\overline{X_2}] + 2COV[\overline{X_1}, \overline{X_2}]\Big) \\ &= 1/2\Big(Var[\overline{X}] + COV[\overline{X_1}, \overline{X_2}]\Big) \end{aligned}$$

Since $\overline{X_1}$ and $\overline{X_2}$ are, by construction, negatively correlated, the variance would be reduced.

However, in addition to the requirement in terms of implementation (production of a couple of antithetic sequences of r.v.), it is especially necessary to get a *perfect synchronization* between the two sequences. Thus, this technique is much more delicate to implement than the aforementioned CRN method.

5.6. Conclusion

In conclusion, we recall that a study by simulation is primarily a statistical study (whereas the construction of a simulation model is primarily relevant to computer-programming). Thus, it is necessary to respect the rules and practices of the statistics. A particular precaution to be taken in a simulation, compared to a more classical statistics problem (for example, an opinion poll), is that simulation handles stochastic processes of which it is necessary to secure the stationarity and to eliminate the transient period from it. In addition, the outcomes cannot be considered as independent.

Thus, it is necessary to choose an adequate duration of simulation, as well a suitable data collection method. The guideline is to obtain estimates of the physical quantities (often in the form of mean values) with a quality (measured through the confidence interval) which is compliant with the aims of the study.

6

OMNeT++

In this chapter we present the discrete event simulation environment OMNeT++. Its modular architecture, as well as the availability of numerous packages and models specifically designed for computer networks, make it suited for modeling computer networks of various natures. In this chapter, we use it primarily to illustrate, through concrete examples, the general concepts presented in the preceding chapters. OMNeT++ indeed constitutes a good support for these illustrations, because of its conceptual philosophy, of its architecture as well as of its availability as an *open source* software.

6.1. A summary presentation

OMNeT++ is a software development environment which is devoted to discrete events simulations. OMNeT++ stands for *Objective Modular Network Testbed in C++*. Being based on the language C++ and adopting a modular approach, OMNeT++ offers a general framework for simulating any discrete system. With its numerous pre-built modules for computer networks, and more particularly for Internet technology as well as mobile networks, it is well suited for simulation of computer networks. It is an *open source* software with a licence policy which is similar to the well-known GNU General Public License (GPL), and, as is, supported by its many contributors. Its popularity is in constant growth both in academic and industrial world. Dr. András Varga conceived OMNeT++ and always ensures its development. All useful information (licence, distribution and documentation in particular) are available on the official Website www.omnetpp.org. The wiki rubric of the latter also contains useful information.

Before continuing, the author wishes to recall and emphasize the objectives of this chapter: to illustrate the general concepts presented in the

Performance Evaluation by Simulation and Analysis with Applications to Computer Networks, First Edition. Ken Chen. © ISTE Ltd 2015. Published by ISTE Ltd and John Wiley & Sons, Inc.

preceding chapters through concrete examples carried out under OMNeT++. This one indeed constitutes a good support for these illustrations, because of its conceptual philosophy, its architecture and its availability. It is in no case a kind of "tutorial" of OMNeT++. The documentation which comes with the software (in particular the tictoc example), as well as that provided by the OMNeT++ site, fulfils this role perfectly. The few indications of an operational nature given by the author simply aim at facilitating the understanding of examples presented in this book without necessity to master the whole OMNeT++ language.

6.2. Installation

OMNeT++ is available under various operating systems: Windows, Linux, Mac OS (precise information is on the official site). In the remainder, by default, we will use the version[1] V4.5 (published on 16/07/2014), under the Windows environment.

6.2.1. *Preparation*

OMNeT++ makes use of the Java language. We must ensure the availability of the latter on the computer (the Java program is located under Program Files\Java). For installing OMNeT++, simply follow the next steps:

– download the current version (omnetpp-4.5-src-windows.zip);

– extract the source from this file which produces the directory omnetpp-4.5;

– place this one under a directory whose name does not contain space, for example C:\MyOMNeT (note: do not put it under Program Files, due to space).

In the remainder of this section, we will present this directory as the *root* during the installation of OMNeT++. All files of OMNeT++ will be found under this directory, including official documentation (under \doc, in particular the pdf files InstallGuide, User Guide, Manual).

1 The most up-to-date version is V4.6 which is published on 02/12/2014 that we have not yet tested.

6.2.2. *Installation*

The OMNeT++ distribution comes with *source* programs which have to be configured and compiled before the first use. The installation of OMNeT++ is done through command-line interface. It is necessary to launch first a command-line window. In this window, it is necessary to be placed under the root directory for launching mingwenv. This ensures we are within the OMNeT++ environment. We have to run:

– ./configure;

– then, make for creating the OMNeT++ environment (*this operation may take some time*).

It is then possible to launch the graphic environment via the command omnetpp. It is also possible to launch the graphic environment as a classical Windows program by launching the program omnetpp.exe under the subdirectory IDE.

6.3. Architecture of OMNeT++

OMNeT++ is a modular language facilitating the modeling of very complex systems by assembly of the more elementary modules. This modular approach is carried out on two levels:

– construction of basic modules using the C++ language. OMNeT++ proposes the two following types:

- simple module which is built with C++ classes, facilitating in this way an algorithmic description of rather complex structure and/or behavior;

- channel which ensures the connection between the modules;

– creation of a more complex compound module by assembly of existing modules, using a language specific to OMNeT++, the *Network Description* (NED). Due to this, it is possible to create more complex systems.

Thus, a simulation model, referred to as network, can be built by interconnecting modules (either simple or compound) using channels (either basic or enhanced), with this hierarchical organization of the components:

– simple module;

– channel;

– compound module;

– network.

This approach facilitates the modeling of very complex systems, whilst nevertheless being able to describe any of its component in its least details if needed.

In addition, OMNeT++ offers a flexible mechanism for the definition of the formats of message. This is particularly useful for the simulation of computer networks.

6.3.1. *Simple module*

The *simples modules* constitute, along with the *channels*, the basic building blocks of OMNeT++. A simple module is configurable through its parameters; it interacts with the other modules through gates. It is implemented in C++ and is described macroscopically by the NED language (see section 6.7, in particular Listing 6.1 and Listing 6.2, for the first examples).

Every simple module is derived from a basic class (cSimpleModule). The latter contains generic (virtual) methods which must be specified for each simple module according to its behavior. The principal methods are:

– initialize and finish which are called, respectively, at initialization (just before the start of the simulation) and after the end of simulation;

– handleMessage which is the *kernel* method through which the behavior of the module is implemented. A module adopting this method acts as a *finite states machine* (FSM) responding to *stimuli* which arrive under the form of *messages* (class cMessage);

– There is also a method called activity, which is the dual method of handleMessage. Basically, a module adopting this method acts as a *process* waiting for requests (for instance, a server waiting for connections). The authors of OMNeT++ recommend to prefer the method handleMessage.

6.3.2. *Channel*

A `channel` allows us to establish links between modules, both for simple modules and compound modules. A channel can be considered as a special case of the simple modules. It is described through parameters like time, rate error and flows; it can be unidirectional (input or output) or bidirectional.

6.3.3. *Compound module*

A `compound module` is built up by assembling modules (simple and/or compound) which are connected by `channels`. A `compound module` has its own `gates` and `parameters`. The modules (simple and/or compound) used for the composition of a compound module are listed under `submodules`. The connections between them are specified under `connections`. An example is given in the section 6.10.3.

6.3.4. *Simulation model (*`network`*)*

The model of the whole system is defined under the key word `network` with the same construction syntax as the one used by `compound module`. Thus, it can be viewed as the single biggest "compound module". By definition, it does not have `gates`, as it is about the system as a whole.

Every simulation is run over a `network`. Thus, within any `OMNeT++` project, there must be at least:

– definition of one `network` through a description (`ned`) file;

– one `network` which is referred as part of the simulation configuration in an initialization (`ini`) file.

6.4. The NED langage

The NED Language is specifically designed for `OMNeT++` to provide a macroscopic description of a module (`gates`, `parameters`, etc.). This description is carried out in the form of a textual file (with the suffix `ned`). We will present the features of NED that we use in our examples at their first occurrence; this should cover most of the basic functions of NED. A complete presentation of NED is out of the scope of this book. For more details, please consult the `User Guide`.

6.5. The IDE of OMNeT++

OMNeT++ offers an *Integrated Development Environment* (IDE) which is based on *Eclipse*. Most of the operations (programming, configuration, simulation running, data display and analysis, etc.) can be done under this IDE:

– to create a project, just choose New/Project then OMNeT++ project. There are two options: an empty project or a project containing two subdirectories src and simulations;

– then, we have to create at least one network, through New/Network or New/Network description File (ned), by using the NED editor. At any time, we can switch between the graphical mode (design) and the text mode (source). Under the NED editor, there is a palette facilitating the creation of various modules (simple, compound, channel, etc.). It gives access to all the modules of currently openned projects. In particular, if the *project* inet is open, we have access to all of the Internet components which are defined in inet. We can select a module by a double-click, then place it into the module under edition (under design mode, of course);

– once the definition/programming of modules and the network is finished, the project has to be built via compilation through Project/Build Project;

– before launching the simulation, we have to configure it by creating an *initialization* file (with suffix ini) through New/Initialization File (ini). This file has to indicate at least one ned file containing the description of a network;

– the launching of a simulation under IDE is done by clicking on a file with suffix ini (generally omnetpp.ini). It is also possible to launch simulation under command line mode; this option is efficient when we launch a set of simulations;

– for the graphical analysis of output data, we have to create a file with suffix anf through New/Analysis File (anf).

6.6. The project

In OMNeT++, a project represents a system to be simulated. Each project is located in a specific directory and contains one or more models. The name of the project is that of the directory, in lower case letters.

A project can be a stand-alone system, such as for example, an $M/M/1$ queue. It can also be used by other projects with the mechanism of `import`. For instance, we can define a project to study the behavior of a single queue (such as an $M/M/1$), then the module modeling this single queue can be *imported* by other projects who need this module.

6.6.1. *Workspace and projects*

`OMNeT++` uses the vocabulary of `workspace` to refer to *the* directory which gathers all of the `projects`. Thus, at the launching of the IDE, the first task is to set a `workspace`. It is possible to change workspace thereafter. This change causes a reinitialization of the environment IDE.

6.6.2. *Creation of a project*

The creation of a project is carried out via `New/Project`. Let us take the example of a project named `MyBasic` which is created with option `empty` project. The IDE will create a directory named `mybasic` with, inside, a file named `package.ned`. This file contains a line which is `package mybasic`. For a first example, just inhibit this line by turning it as a *comment* line. A comment line in `NED` is a line starting with `#`. We will come back to the meaning of `package` later (see section 6.10.2.1).

More usually, we should use the alternative option to create a project with two subdirectories:

– `src` where the source programs are located;

– `simulations` where the configuration and initialization files are located.

6.6.3. *Opening and closing of a project*

We proceed to the *opening* (respectively *closing*) of a project by selecting it in the `Project Explorer`. There are then two possibilities of opening (respectively closing):

– via `Project` then `Open Project` (respectively `Close Project`);

– directly via the drop-down menu by the right click.

6.6.4. *Import of a project*

The *import* functionality allows us to incorporate the modules of a `project` into another one. The latter can then exploit these modules like its own. This is an effective way of capitalizing the existing modules by facilitating their reuse. The import is carried out through `Import/General/Existing Project into Workspace`.

For example, we can import in this way all the subsystem `inet` which gathers a significant number of modules simulating different components of the Internet.

6.7. A first example

In this section, we present a first example of `OMNeT++`. We choose to make this example as simple as possible, but nevertheless contain the basic functionalities of `OMNeT++`. It is composed of a source generating a sequence of messages toward a fixed destination. This example is deprived of all the auxiliary functionalities, those related to the presentation and the data-gathering in particular. This model will be used as a basis to which we will gradually add other functionalities to illustrate the many possibilities of `OMNeT++`.

6.7.1. *Creation of the modules*

In this example, the system contains exactly two functional blocs, a *source* named `MySource` and a *destination* named `MySink`. Each bloc is a `simple module` in the sense of `OMNeT++`. Each simple module is jointly defined via a description file in NED language `.ned` and an implementation file in C++ language (`.cc`). The `C++` file comes normally with the companion `.h` file. Here `.h` file is absent for the sake of simplicity, this example is actually extremely simple.

6.7.1.1. *The module* `MySource`

This module is described macroscopically in the file `MySource.ned` and is implemented in the `MySource.cc`.

6.7.1.1.1. Description

In the description file (see Listing 6.1), we specify the following elements:

– this module has an unidirectional `gate`, in the `output` sense, which is named `out`. This is actually the interface through which messages will be sent out;

– the traffic rate can be configured via a `parameter` referred to as `sendIaTime`. It represents the duration of the inter-arrival interval. Its unit is in second (s), this is indicated by `@unit(s)` (more generally, for NED, every thing begins with a `@` is a `property`; here, we have the *unit* property).

```
simple MySource
{
  parameters:
    volatile double sendIaTime @unit(s);
  gates:
    output out;
}
```

Listing 6.1. *Description in* `MySource.ned`

6.7.1.1.2. Implementation

The implementation is given through the *homonym* file with `cc` extension (see Listing 6.2). We build a source generating messages which are separated by `sendIaTime` in time. Since `OMNeT++` is event-driven, the sending of a message should be an event. This event is materialized by the mechanism of *self-message*, i.e. an auxiliary message that the module sends to itself, in order to create an event. Here, the *self-message* is named `msgSend`. This message "wakes up" the source module. The latter proceeds to create (`new`) a new message (`msg`) then sends it out through the gate `out`. The next event occurrence time of the self-message `msgSend` is then programmed through the method `scheduleAt`. In this way, the source generates messages after messages with an interval of `sendIaTime`. Hereafter, you can find some additional comments on Listing 6.2:

– every new `simple module` is derived from the basic class `cSimpleModule`;

– we must always use `Define_Module` to declare a new module;

– we define the following two virtual functions, `initialize`[2] and `handleMessage`:

- `initialize` performs initialization just before the start of simulation. Here, its sole role is to launch the process of *self-message* by creating the self-message then programming its first occurrence;

- `handleMessage` is a method which takes place in every `simple module`. It is basically through the programming of this one that we define the behavior of a module. Here, this behavior simply consists of generating then sending a (real) message, before scheduling the next event.

```
#include <omnetpp.h>
class MySource : public cSimpleModule
{
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msgSend);
};
Define_Module(MySource); // MANDATORY!
void MySource::initialize()
{
    // Start the process with this self-message
    scheduleAt(simTime(), new cMessage);
}
void MySource::handleMessage(cMessage *msgSend)
{
    // generate a new message
    cMessage *msg = new cMessage;
    // send this msg to the gate "out" (see ned)
    send(msg, "out");
    // program the next send event "sendIaTime",
    // sendIaTime is a parameter defined in ned
    scheduleAt(simTime()+par("sendIaTime").doubleValue(),
    msgSend);
}
```

Listing 6.2. *Implementation in* `MySource.cc`

2 We will show the usage of its dual function `finish` later (see Listing 6.7) in an advanced version of the module `MySink`.

6.7.1.2. *The module MySink*

This module plays the role of the destination. We can notice that its description (see Listing 6.3) and its coding (see Listing 6.4) are very simple. Actually, it has one `input gate`, and the sole action invoked by each event (arrival of a message) is the destruction of the message. However, its role is *essential*. Indeed, the message, which arrives at the end of its *journey* and has no more utility, has to be destroyed, in order to release the memory space it occupied. Otherwise, there would be potentially an enormous waste of memory capacity, should the simulation run over a long period.

```
simple MySink
{
    gates:
      input in;
}
```

Listing 6.3. *Description in* `MySink.ned`

```
#include <omnetpp.h>
class MySink : public cSimpleModule
{
  protected:
    virtual void handleMessage(cMessage *msg);
};
Define_Module( MySink );
void MySink::handleMessage(cMessage *msg)
{
  delete msg; // MANDATORY: otherwise memory waste!
}
```

Listing 6.4. *Implementation in* `MySink.cc`

6.7.1.3. *The simulation model*

Now, we can create the model (`network`) through another `ned` file (see Listing 6.5). In this file, we specify its components (`submodules`), here `MySink` and `MySource`, as well as their relations (`connections`), here a simple unidirectional link (through the key word `-->`).

We define in this way a `network` referred as `MyBasic`. This one is built with two modules `source` and `sink`, with the `out gate` of `source` being linked unidirectionally (`-->`) to the `in gate` of `sink`.

```
network MyBasic
{
    submodules:
        source: MySource;
        sink:   MySink;
    connections:
        source.out --> sink.in;
}
```

Listing 6.5. *Composition of the model (*`MyBasic.ned`*)*

6.7.2. *Compilation*

We launch the compilation via `Project/Build Project`. If everything is in order, this action produces a runnable file named `mybasic.exe`. In the case of errors, messages will appear in a way which facilitates their identification and correction.

6.7.3. *Initialization*

Before running the simulation, it is still necessary to fix the initialization condition through a file with suffix `ini`, generally, we name it `omnetpp.ini`. Such a file is given below (see Listing 6.6) with some comments as follows:

– the model to be simulated is specified in the `MyBasic.ned` file (through the key word `network`);

– simulation stops (`sim-time-limit`) at the end of 10 s of simulated time;

– there are two possible scenarios (`[Config]`): `BasicLow` and `BasicHigh`, the choice will be made at the launching of the simulation;

– the nature of the parameter `sendIaTime` deserves special attention. We may notice that it is *more than* a simple parameter, it is in fact a *function*, and more precisely, a function generating a *random variable* (here it has an exponential distribution). Each time the `source` module makes use of this parameter, a random variable is generated. It is interesting to notice that we can change, through this configuration file, the rate as well as the distribution. This is a rather judicious feature, since it allows us to testing various arrival pattern. Here, we have a Poisson process. The double `**` means that the definition is valid for any instantiation of the module `source` (we have only one in this example).

```
[General]
# The systm to be simulated
network = MyBasic
# the finish time (in simulated time)
sim-time-limit = 10s
# Simulation scenario with low arrival rate
[Config BasicLow]
# The title of the scenario
description = "low job arrival rate"
# definition of the parameter sendIaTime:
# a random variable following expotential law
# mean value : one arrival every 2 sec.
**.source.sendIaTime = exponential(2s)
# Another scenario with higher arrival rate
[Config BasicHigh]
description = "high job arrival rate"
**.source.sendIaTime = exponential(1s)
```

Listing 6.6. *Initialisation file:* `omnetpp.ini`

6.7.4. *Launching of the simulation*

A session of simulation can be launched in two ways:

– through the *command line* interface, we have to go to the project's directory then invoke `mybasic.exe`;

– under the IDE, while clicking on the file `omnetpp.ini` or while passing by the menu `run`, then to choose "`run as`" then the item `OMNeT++` simulation.

6.8. Data collection and statistics

The preceding example is voluntarily reduced to its extreme simplicity, in order to show the most elementary functionalities of `OMNeT++`. This simulator will run as an ordinary computer program. However, it is absolutely useless as a simulation program, since it does produce any data for analytical purpose. We introduce hereafter some elementary functionalities for data collection and statistical analysis.

In the real world, when we want to observe a phenomenon (for example the temperature of a room), we put a probe (a thermometer in fact). In `OMNeT++`,

we can install a "probe" in two ways: via the `Signal` mechanism or by using the statistics collectors.

All results of each simulation (`run`), either by the `Signal` mechanism or by the collectors, will be arranged in two common files, one for the scalars (suffix `sca`) and the other for the time series (or vectors, suffix `vec`).

6.8.1. *The* `Signal` *mechanism*

In the `OMNeT++` architecture, the `Signal` mechanism permits the capture of a time-indexed sequence of information related to a quantity called signal: $S = \{x_t\}$ where x_t is the information at the moment t of the signal S. x_t is captured via the function `emit(S, v)`, t is *implicitly* the (simulated) instant of the invocation of `emit`, and `v` is the value of x_t to be captured. An example of signal can be the size of a queue which is programmed to be captured upon each arrival and departure.

In its current state of development, captured information is mainly of the scalar type, but in the general concept of `OMNeT++`, they can take a richer form. This mechanism is thus a generic tool; its use is not limited to the statistics. Here, we examine it only through the view point of the statistics.

A signal is attached to the module in which information is taken. A signal, say, `abc`, is created in 2 steps:

1) declaration of a (`private`) variable receiving the identifier of the signal:

```
simsignal_t abcSignal;
```

where `abcSignal` is the identifier (of type `simsignal_t`) of the signal `abc`;

2) register the (`registerSignal`) signal in `initialize` method:

```
abcSignal = registerSignal("abc");
```

REMARK.– the dual function `getSignalName(abcSignal)` returns "abc".

At any moment within a simulation, it is possible to emit information d related to the signal `abc` by `emit(abcSignal, d)`. This process is also open to every module, in the sense that every module can *subscribe* to a given signal and thus be able to gain access to it.

Obtaining various types of statistics (mean, histogram, etc.) from a `Signal` is made through the property `@statistic[x]` that one specifies in NED file of the module. Taking the signal `abc` of our example, it will be `@statistic[abc]`.

6.8.2. *The collectors*

`OMNeT++` provides also a more traditional approach of statistics gathering via collectors. One of the most useful collectors is the `cStdDev` which provides standard statistics (mean, variance, extremities, etc.). It is also possible to obtain a time series with the collector of type `cOutVector`.

To obtain the standard statistics of a variable, say `abc`, the three essential steps of this approach are to:

1) declare the collector with:

```
cStdDev  abcScalar;
```

2) collect the data (usually done in the `handleMessage` method) with:

```
abcScalar.collect(d);
```

3) record the results (usually at the end of simulation through the `finish` method):

```
abcScalar.record();
```

This approach is less flexible than the `Signal` mechanism. Indeed, each type of statistics must be taken by a suitable collector. If we want to obtain a histogram, in addition to the standard statistics, we have to declare a second collector of type `cHistogram`. Moreover, these choices have to be firmly coded into the module, whereas with `Signal` mechanism, one module can subscribe to, then unsubscribe to, a signal at wish.

6.8.3. *Extension of the model with statistics*

We extend the previous example `MyBasic` with statistics on sojourn time (`lifetime`) of the messages, by using both of the two approaches.

This quantity can only be measured at the end of life of a message, thus, the probe has to be attached to the terminal module `MySink`. As a consequence, the extensions concerns solely this module (both on `cc` file and `ned` file), with no impact on `MySource`.

6.8.3.1. *Put probes in* `MySink.cc`

We will put a signal entitled `lifetime` as well as a traditional collector entitled `lifetimeStdDev` (see Listing 6.7). The main comments are:

– creation of a signal named `lifetimeSignal` as well as a collector of the type `cStdDev`;

– the declaration of `lifetimeSignal` in `initialize`. We include there also the *naming* of `lifetimeStdDev`, as it is recommended by `OMNeT++`, even if that is not functionally necessary for our program;

– their use in `handleMessage`: before destroying a message, we measure its sojourn time (`d`) in the system (the creation time of the message is a generic attribute that can be obtained via `GetCreationTime`), then send (`emit`) and `collect` the obtained measurement `d`;

– the explicit recording of results of `lifetimeStdDev` in `finish()` and, for `lifetimeSignal`, the choice of the statistics in the `ned` file (see Listing 6.8).

Hereafter is the new version of `MySink.cc` :

```
#include <omnetpp.h>
class MySink : public cSimpleModule
{
  private:
    simsignal_t lifetimeSignal; // id of the signal
    cStdDev     lifetimeStdDev; // collector
  protected:
    virtual void initialize();
    virtual void finish();
    virtual void handleMessage(cMessage *msg);
};
Define_Module(MySink);
void MySink::initialize()
{
    lifetimeSignal = registerSignal("lifetime");
    lifetimeStdDev.setName("Lifetime");
```

```
}
void MySink::finish()
{
    recordScalar(" Lifetime's Std Stats ", simTime());
    lifetimeStdDev.record();
}
void MySink::handleMessage(cMessage *msg)
{
    // get the sojourn duration (lifetime)
    simtime_t d = simTime() - msg->getCreationTime();
    emit(lifetimeSignal, d); // signal
    lifetimeStdDev.collect(d); // collector
    delete msg;
}
```

Listing 6.7. *Data collection in* `MySink.cc`

6.8.3.2. *Configuration through* `MySink.ned`

This operation is related only to the `Signal` mechanism. Here, we establish from the data of the signal `lifetime` three statistics (see Listing 6.8):

– two `scalar` quantities: `mean` and `max`;

– all values of the signal (the time series) in the form of a `vector` with two dimensions: the value and the moment of its emission.

```
simple MySink
{
  // declare the lifetime signal (with type),
  // Recommanded but Currently NOT used
  @signal[lifetime](type="simtime_t");
  // record both vector  and scalar (mean, max)
  @statistic[lifetime](title="lifetime"; unit=s);
  @statistic[lifetime](record=vector, mean, max);
  gates:
    input in;
}
```

Listing 6.8. *Selection of statistics in* `MySink.ned`

6.8.4. *Data analysis*

The data are recorded in the subdirectory `results` of the project. They are identified by the name of the scenario of simulation. They are ASCII files and can thus easily be converted and/or processed by various statistical tools.

`OMNeT++` offers a graphical analysis tool under IDE. To invoke it for the first time, choose `File/New/Others`, then the `wizard` for `Analysis file (anf)` under the rubric `OMNeT++`. We can also click directly on the result files. We get a file with `anf` suffix. There are various operations that can be carried out on `anf` files (see `User's Guide` for details).

6.9. A FIFO queue

We present here a more advanced example, through a first in first out (FIFO) queue, which includes in particular the handling of the message exchange between modules. We also enhance the `MySource` module with a richer behavior. We present especially some useful practices through the configuration of different simulation scenarios.

6.9.1. *Construction of the queue*

The main add-on compared to the preceding examples is the presence of a queue which consists of a waiting room, i.e. a *passive* storage entity, as well as a server. Storage is coded using `cQueue` which is a predefined class in `OMNeT++`.

6.9.1.1. *Description*

The description of a FIFO queue is very simple, it is about an entity which has an input and an output. Since it incorporates a service mechanism, we add a parameter specifying the service distribution. Listing 6.9 specifies this queue, with additional property related to the statistics (already presented in the preceding section).

```
simple MyFifo
{
  parameters:
    volatile double ServiceTime @unit(s);
    @signal[qLength](type="int");
    @statistic[qLength](title=" Queue Length");
```

```
    @statistic[qLength]( record=vector, max, mean);
  gates:
    input in;
    output out;
}
```

Listing 6.9. *Description of* `MyFiFo`

6.9.1.2. *Insertion of a message*

The messages (class `cMessage`) are generated by the source to be sent toward FIFO queue. The *event* "arrival of a message" leads to the insertion of the message in the tail of the queue by `insert`, which is a standard method of the class `cQueue`.

6.9.1.3. *Service*

The implementation of a service is a little less straightforward than that of the insertion of a message. Indeed, we deal with a discrete event simulator. The insertion of a message is automatically invoked by the arrival event, whereas the starting of a *new* service is an action which is *conditioned* by the presence of at least a message as well as the availability of the server. It is only after a positive checking of these two conditions that an event of service can, and will, be programmed. There are two sequences of events leading to such a checking:

– upon arrival of a message;

– at the end of the current service: this event is created via an artificial `self-message` (namely `msgEoS`, see Listing 6.10). The sending of `msgEoS` is scheduled at the beginning of the current service with the knowledge of its duration.

6.9.1.4. *Sending of a message*

At the end of each service, the currently finished customer (modeled here by a message) must leave the queue to be sent toward `MySink`. As the basic class stored in `cQueue` is `cObject` and not `cMessage` (this one is actually derived from `cObject`), we need to "retype" the object as `cMessage` before sending it.

```
msgToSendOut = (cMessage *) myQueue.pop();
send(msgToSendOut, "out");
```

6.9.1.5. *Statistics*

We inserted two probes:

– `qLength` (queue length), via the `Signal` mechanism;

– `busyStdStat`, a classical collector which observes if the server is busy.

The two statistics are taken with each event. They are added to those already defined in the `MySink` module.

6.9.1.6. *The FIFO queue model*

In this way, the FIFO queue is implemented here as an entity which is activated by message arrivals as well as by the ends of service. The comments are within the code (Listing 6.10). Due to page formatting limitation, some of the code lines in this listing, as well as some following ones, may be split and/or reformatted.

```
#include <omnetpp.h>
class MyFifo : public cSimpleModule
{
  private:
    // a PASSIVE message storage entity
    cQueue   myQueue;
        // self-msg for  end of service (EoS)
    cMessage *msgEoS;
        // for poped out msg from queue
    cMessage *msgToSendOut;
        // 0 = not busy
    int    serverBusy;
        // qLength as signal
    simsignal_t qLengthSignal;
        // get busy ratio via collector
    cStdDev  busyStdStat;
        // for transient period
    simtime_t runWarmupPeriod;
  protected:
    virtual void initialize();
    virtual void finish();
    virtual void handleMessage(cMessage *msg);
};
Define_Module(MyFifo);
void MyFifo::initialize()
{
```

```
    serverBusy=0;
    qLengthSignal = registerSignal("qLength");
    runWarmupPeriod=simulation.getWarmupPeriod();
    msgEoS = new cMessage("End of Service");
}
void MyFifo::finish()
{
    recordScalar("Busy ratio", simTime());
    busyStdStat.record();
}
void MyFifo::handleMessage(cMessage *msg)
{
  if (msg == msgEoS) {// either EoS or a new customer
     //basic unit in Queue=object, retyped to msg
     msgToSendOut = (cMessage *) myQueue.pop();
     send(msgToSendOut, "out");
     serverBusy=0; // Server free again
  }
  else { // new msg, insert to  the queue
      myQueue.insert(msg); // insert: to tail(Back)
  }
  // Check Service opportunity at each event
  if ( (serverBusy==0) and (myQueue.length()>0) ) {
    serverBusy=1; // Service begins
    double t=par("ServiceTime").doubleValue();
    scheduleAt(simTime()+t, msgEoS);
  }
  // Take  statistics at each event
  emit(qLengthSignal, myQueue.length());
  if (simTime()>runWarmupPeriod) {
    busyStdStat.collect(serverBusy);
  } // Warmup for collector must be hard-coded!
}
```

Listing 6.10. *Implementation of* `MyFiFo`

6.9.2. *Extension of* `MySource`

We made the following extensions to the source (`MySource`):

– we can from now on send a *burst* of messages at the first transmission opportunity, this makes it possible to simulate various initial conditions;

– moreover, the moment of the first emission is from now on a parameter.

The details are found in Listing 6.11, for its description, and Listing 6.12, for its implementation. We could notice this *cosmetic* detail (`@display`) which allows us to display an image "source" for the modules of `MySource` type.

```
simple MySource
{
  parameters:
    int      InitialBurst = default(1);
    double   InitialDelay @unit(s) =default(0s);
    volatile double sendIaTime @unit(s);
    @display("i=block/source");
  gates:
    output out;
}
```

Listing 6.11. *New description of* `MySource`

```
#include <omnetpp.h>
class MySource : public cSimpleModule
{
  private :
    int remainInitialBurst;
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msgSend);
};
Define_Module(MySource);
void MySource::initialize()
{
    remainInitialBurst=par("InitialBurst").longValue();
    // line too long, split into 2:
    double t=par("InitialDelay").doubleValue();
    scheduleAt(simTime()+t, new cMessage);
}
void MySource::handleMessage(cMessage *msgSend)
{
  cMessage *msg = new cMessage("Customer");
  send(msg, "out");
  if (remainInitialBurst>1) {
    remainInitialBurst--;
    // burst=next event at the same instant
    scheduleAt(simTime(), msgSend);
```

```
  }
  else { // usual case (after initial burst)
    double t= par("sendIaTime").doubleValue();
    scheduleAt(simTime()+t, msgSend);
  }
}
```

Listing 6.12. *New implementation of* `MySource`

6.9.3. *Configuration*

For the sake of simplicity, we do not present the aforementioned basic part of the configuration of the model (`network`), in order to focus on some current practices of simulation. We illustrate them through following functionalities that we can specify at the configuration step (through the `.ini` file, see Listing 6.13):

– declaration of 2 PRNG ($U(0,1)$ streams) (`num-rngs=2`) and association between a generator of a module and one of these 2 streams (see section 3.4.5.3):

- the arrival process (generated in module `source`) will be generated with the stream 0 (`**.source.rng-0=0`);

- the service time (generated in module `fifo`) will be generated with the stream 1 (`**.fifo.rng-0=1`).

– parametric launching of several simulations (can only be done in *batch* mode under *command line* environment) with the following syntax: `${i,j, k}`. In this example, there are two values for initial customers (1 and 10) as well as 4 different cases for the service distribution, thus there is a total of 8 simulations scenarios;

– replication: it is possible to launch several simulations for the same set of parameters (but, of course, different seeds, see below) via the command `repeat`. Here, we have three replications. Thus, we have a total of 24 simulation runs.

– choice of the *seed*: by default, `OMNeT++` ensures the choice of the seeds. It is possible to specify our own choices. To specify the seed related to stream 1 of a PRNG MT for example, the instruction is `seed-1-mt`. In our example, we specified for each of the three repetitions, the seed for each of the 2 streams:

the values are given in an ordered manner, i.e. the 3 couples of the seeds are $(11, 19)$, $(23, 41)$, $(45, 77)$;

– the transient period (see section 5.2.4) can be specified through the instruction `warmup-period`. Here, it is set to 100 s, i.e. 5% of the total simulation duration. Since this period must be the same for all replications, we put it in the `[General]` section. It is to be pointed out that the transient period is automatically respected by the `Signal`, by a built-in mechanism of `OMNeT++`. On the contrary, for the data collected through `collector`, a hard coding[3] is required for each individual probe. It is what we did in this example.

```
[General]
network = MyExamples
record-eventlog = true
sim-time-limit = 2000s
# warmup: transient period elimination
warmup-period=100s
# Two RNG for arrival and service
num-rngs=2
**.source.rng-0=0
**.fifo.rng-0=1
# Parametric simulation scenarii
[Config C1]
description = "Parametric runs"
**.source.sendIaTime = exponential(4s)
**.fifo.ServiceTime = exponential(${1, 2, 3, 3.5}s)
# Initial conditions
**.source.InitialBurst = ${1, 10}
# repetition and seed setting
repeat=3
seed-0-mt=${11, 23, 45 ! repetition}
seed-1-mt=${19, 41, 77 ! repetition}
```

Listing 6.13. *Parametric configuration of simulation runs*

3 Thus, it can be a semantic bug if we forget to hard code it for a probe: this would lead to a lack of accuracy of the statistics and this bug may not be discovered.

6.10. An elementary distributed system

6.10.1. *Presentation*

We present here a complete example, built from the scratch, modeling an elementary distributed system (here a *Token Ring*), as an illustration of the functionalities of `OMNeT++`. We can find most of them, under various forms, in other advanced simulation tools. *Token Ring* is chosen since it is one of the most fundamental protocols in distributed systems (see e.g. [LeL 78]). We take the version of *Token Ring* which ensures the regeneration of the token in cases of its loss.

Through this example, we aim to show how to model a distributed system, to which belong the computer networks. We will also show how to build hierarchically a more complex system from basic modules.

The Token Ring algorithm consists of organizing entities as a virtual ring where a single right of action, called *token*, circulates among the entities by following the virtual ring. Only the holder of the token can start an action (transmission, access to a data base, etc.). As such, the algorithm is vulnerable, because the loss of the token leads to the fatal stop of the whole system. Various mechanisms of regeneration were considered. We consider here the following mechanism:

– we set up a process for a periodic checking of the token's presence;

– upon each passing of the token, its presence is reconfirmed via a flag;

– in the event of loss, a particular alarm message (*Claim Token*) is sent by the entity to all of the entities on the ring;

– in the event of conflicts, i.e. (almost) simultaneous sending of *Claim Token* message by several entities, which is a completely plausible possibility, the priority is given to the entity having the (numerically speaking) largest identifier.

We will implement such a distributed algorithm in the `TRNode` modules. As the distributed algorithms work with exchange of specific messages, we also introduce a specific format of messages (`MyPacket`), due to a functionality of `OMNeT++` (see section 6.10.2.2). We recall that a communication protocol is a distributed algorithm, the messages exchanged between the entities are included in the header of the Protocol Data Unit (PDU).

We summarize below the principal functionalities of `TRNode`, as well as the attributes of the messages:

– we define a particular format of messages containing the identity of the transmitting entity as well as the type of the message. This last is either `Token` (free token) or `ClaimToken` (alarm);

– we distinguish two processes which are held in parallel on `TRNode`: the normal mode (circulation of the token) and the control of the token's presence:

- the first one is related to the events caused by the arrival of a `myPacket` type message;

- whereas the second is materialized by events caused by the arrival of a `self message` of the `msgCheckToken` type.

– the process "control of the token's presence" is activated periodically (`CheckTokenInterval`). At each passage, one of the two cases happens:

- if the process notices the loss of the token (via the flag `isTokenLost`), it generates a `Claim Token` message with the node's identity as transmitter (Source, `SA`) and enters the *claim-token* mode (`hasClaimedToken` set to `true`);

- otherwise, it sets the flag `isTokenLost` to `true` by anticipation of a loss. If the token exists, this flag will be repositioned to `false` upon the very next passage of the Token. The process then schedules its next passage before leaving.

– the case of the events with the messages of type `MyPacket` is more complex, because these messages can be a `Token` or a `ClaimToken`. Moreover, in case of a `Token`, this one can be simply released (no activity) or it can be held (activity) until the end of the activity before being released. We also notice that the token passing time cannot be neglected (set to zero), even if it is rather small related to token holding (activity) periods. As an implementation detail, the *token-holding* is coded by scheduling an *artificial* couple of "leave" and "come-back" for the duration of an activity (`hodlingDuration`);

– the loss of a message can take place on each arrival of a `Token` message, this causes the loss of the token: indeed, there is only one Token on the ring;

– a claim conflict may occur if one node is itself in *claim token* mode (`hasClaimedToken`) and receives a `ClaimToken` message. It is solved by comparing the node's own identifier and the one of the transmitter of the

incoming ClaimToken message (the SA field);

- if the transmitter of the incoming ClaimToken is itself, this is not a conflict, but rather the *end* of the token-claiming process. The node can generate a brand-new token;

- if the transmitter of the incoming ClaimToken is different, it is actually a conflict. It is solved according to the numerical value[4] of the identifiers: if the current node has larger identifier, it maintains in claim-token mode and destroy the incoming message; otherwise, it exits from the claim-token mode;

– in order to implement the above operations, we need the following information to be carried in messages:

- SA which specifies the identifier of its transmitter;

- Type which indicates if it is a Token or a ClaimToken.

6.10.2. *Coding*

We briefly present the main points about the coding of this example. Additional details are in the comments incorporated in the listings.

6.10.2.1. *Namespace*

We present through this example the usual manner to develop a project under OMNeT++ with its own *namespace*. The technique of *namespace* is classically used for the modular development of complex systems: each module can be coded in its own space with its own elements with freely chosen identifiers.

In OMNeT++, this is done with the following organization and syntax. We assume that the name of the namespace is tokenring.

– under the root directory of the project (tokenring), two subdirectories are created, respectively src and simulations;

4 It is a common practice in the computer networks. This solution is not without risk: if two entities carry the same identifier, the algorithm does not work any more. We find here the same risks as in real situations (bad assignment of address, for example). We assume here that this does not occur.

– under src, there are all of the modules. The file package.ned now contains the following lines:

```
// designate the NED package of this folder
package tokenring;
// namespace of module C++ classes
@namespace(tokenring);
```

– under simulations, there is the network description file TokenRing.ned as well as the initialization file omnetpp.ini. There is also a file package.ned which contains the following line:

```
package tokenring.simulations;
```

– in the implementations in C++, as we can note through Listings 6.16 and 6.17, the codes from now on are embedded within the *namespace* tokenring.

```
namespace tokenring {
...
}
```

6.10.2.2. *Specific message format*

It is possible to create a specific message format through a file with suffix msg. This functionality of OMNeT++ is quite useful for the modeling of the communications protocols. In our case, it will be the MyPacket.msg file (see Listing 6.14). The OMNeT++ compiler deals with the remainder. The important points are:

– definition of the specific format through a file with suffix msg, say MyFormat.msg;

– the OMNeT++ deals with this file and creates a number of auxiliary files, of which the .h header file with the syntax MyFormat_m.h;

– each module wants to use the message must include the MyFormat_m.h;

– access to the new attributes of the message, within C++ coding, is done through the methods setX() and getX() where X is the name of the attribute. For example, given an attribute named MyId, its two associated methods are setMyId() and getMyId().

We can notice that we can define in the same file literal constants (enum) which will be usable in all the modules C++ which include the .h file. This helps to achieve a clean and auto-documenting coding.

```
enum MyPacketValues
{
  // Type
  Token = 0; // Free Token
  ClaimToken = 1; // Token Recovery
};
packet MyPacket
{
  int SA;  // Source's Id (Address)
  int Type; // 0 = Token, 1=Claim
};
```

Listing 6.14. MyPacket.msg

6.10.2.3. *Coding of the TRNode module*

The detailed coding is given through the description file (see Listing 6.15) and its C++ implementation which is jointly given by the header file (see Listing 6.16) and the C++ coding (see Listing 6.17). Programming-level comments are inserted among the code-lines. We would like to give some general comments as follows:

– we do not aim to get an optimal coding, our goal remains the illustration of elementary functionalities of OMNeT++;

– we intentionally left some codes facilitating the printing of auxiliary messages (the lines EV << xx), in order to get outputs of various informations during simulation run at the very instants where this part of code is running. It is a common practice when developing and/or "debugging" simulation programs.

```
package tokenring;
simple TRNode
{
  parameters:
    //Id: bigger=higher priority,
    int       NodeId =  default(0);
    double    TokenPassingDelay @unit(s)
                          = default(0.01s);
    volatile int TokenHoldIndicator
                          = default(bernoulli(0.2));
    double  CheckTokenInterval @unit(s)
                          = default (50s);
    volatile int LossIndicator
```

```
                              = default(bernoulli(0.1));
    volatile double ActionDuration @unit(s)
                              = default(uniform(1s, 3s));
  gates:
    input in;
    output out;
}
```

Listing 6.15. *Description:* `TRNode.ned`

```
#ifndef __TOKENRING_TRNODE_H
#define __TOKENRING_TRNODE_H
#include <omnetpp.h>
#include "MyPacket_m.h"
namespace tokenring {
class TRNode : public cSimpleModule
{
  private:
    int       NodeId; // this node's Id
    cMessage* msgCheckToken;
    bool      isTokenLost;
    double    CheckTokenInterval;
    double    TokenPassingDelay;
    bool      hasClaimedToken;
    bool      holdingToken;
    double    holdingDuration;
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
    // TR specific functions
    void handleCheckToken(cMessage *cmsg);
    void handleNewPacket(cMessage *nmsg);
};
}; // namespace
#endif
```

Listing 6.16. *Header file:* `TRNode.h`

```
#include <TRNode.h>
namespace tokenring {
Define_Module(TRNode);
void TRNode::initialize()
{
    // get config parameters
```

```
    NodeId = par("NodeId").longValue();
    TokenPassingDelay =
             par("TokenPassingDelay").doubleValue();
    CheckTokenInterval =
             par("CheckTokenInterval").doubleValue();
    // local var initiation
    holdingToken=false;
    isTokenLost = true; // Token to be created
    // waiting return of ClaimToken
    hasClaimedToken = false;
    msgCheckToken = new cMessage("CheckToken");
    scheduleAt(simTime()+CheckTokenInterval,
            msgCheckToken);
} // end initialize
void TRNode::handleMessage(cMessage *msg)
{
  if (msg==msgCheckToken) {// Either CheckToken or Pkt
    handleCheckToken(msg);
  } // end if msg=CheckToken
  else { // It is a Packet
    // emulating lost phenomenon
    if (par("LossIndicator").longValue()>0) {
      EV  << " Loss occurs on node " << NodeId;
      EV << " at" << simTime() << endl;
      delete msg;
    }
    else { // pkt alive
      if (holdingToken) { // End of hold
        holdingToken=false;
        send(msg, "out"); // release token or claim
      }
      else { // it's a Newly Arrived Packet
        handleNewPacket(msg);
      } // else of if holdToken
    } // else of if lost
  } // else of if (msgCheckToken)
} // handleMessage
// the CheckToken process
void TRNode::handleCheckToken(cMessage *cmsg)
{
  if (isTokenLost) {// no change since the last visit
    // generate a claim pkt
    MyPacket* pkt = new MyPacket();
```

```
    pkt->setSA(NodeId);
    pkt->setType(ClaimToken);
    hasClaimedToken=true; // enter in Claim process
    send(pkt, "out");
    EV << "Claim Token sent by node " << NodeId;
    EV << " at " << simTime() << endl;
  }  // if TokenLost
  // Check process=periodic, regardless lost or alive
  // Set to false by the 1st Token passing, if any!
  isTokenLost = true ;
  // schedule the next visit
  scheduleAt(simTime()+CheckTokenInterval, cmsg);
} // handleCheckToken
// Upon arrival of a NEW Packet
void TRNode::handleNewPacket(cMessage *nmsg)
{
  MyPacket* pkt;
  bool      hasDeletedPkt=false;
  pkt = (MyPacket*) nmsg; // access to pkt's fields
  // New Pkt: handle through TR's type
  switch (pkt->getType()) {
    //
    case  ClaimToken : {
      int saIN = pkt->getSA();
      if (saIN<NodeId) { // NodeId higher than saIN
        if (hasClaimedToken) {
          // This Node has a higher RUNNING Claim
          delete(pkt); // delete saIN's claim
          hasDeletedPkt=true; //
        }
        else { // Discover a Claim by saIN
                  // Now, it is becoming  MY Claim
          pkt->setSA(NodeId);
          hasClaimedToken=true;
          EV << "Preempt Claim of node ";
          EV << saIN << " by node " << NodeId;
          EV << " at " << simTime() << endl;
        } // else of if hasClaimed
      } // if (saIN<NodeId)
      else if (saIN==NodeId) { // return of MY claim
        pkt->setType(Token); // Recovery of Token
                 // the end of the claim process
        hasClaimedToken=false;
```

```
        EV << " Token recovered by node " << NodeId;
        EV << " at " << simTime() << endl;
      } // else if ==NodeId
      else { // the case saIN>NodeId: saIN is Higher
            // abandon my claim anyway
        hasClaimedToken=false;
      } // else of if <
      if (!(hasDeletedPkt)) {
        holdingDuration=TokenPassingDelay;
      }
      break;
    } // end ClaimToken
    case Token : {
      if (hasClaimedToken) {
            // abandon, token appears
        hasClaimedToken=false;
      }
      if (isTokenLost) {
        isTokenLost=false;  // Token exists
      }
      // decide if take an action
      if (par("TokenHoldIndicator").longValue()>0)
          {// hold
        double t= par("ActionDuration").doubleValue();
        holdingDuration = t +TokenPassingDelay;
        pkt->setSA(NodeId); //
        EV << " Holding Token by node " << NodeId;
        EV << "  at " << simTime() << endl;
      } // if SendMsg >0
      else { // pass the free Token
        holdingDuration=TokenPassingDelay;
      }
      break;
    } // end Token
    default : {
      EV << "No such Pkt type" << pkt->getType()
          << endl;
      endSimulation();
      break;
    } // default
  } // switch
  // hold the new pkt with correc holdDuration
  if (!(hasDeletedPkt)) {
```

```
    holdingToken=true;
    scheduleAt(simTime()+holdingDuration, pkt);
  } // !hasDeletedPkt
} // handleNewPackt
}; // namespace
```

Listing 6.17. *Implementation:* `TRNode.cc`

6.10.3. *Modular construction of a larger system*

It is important for a simulation language to be able to deal with modular construction, by re-using existing modules, so that we can build complex systems in an efficient manner.

We give an illustration of the modular construction feature of `OMNeT++` through a simple example based on `compound module` (see Listing 6.18). We first create a module referred as `Couple` containing two basic modules `TRNode` which are connected in series. Each `Couple` module has its own `gates`.

Once built, this compound module can contribute to the building of an even more complex module, just like a simple module. This will be illustrated further, when we build the `network` to be simulated with two `Couple` modules and one `TRNode` modules. The three entities form a ring. Actually, a `network` is a particular case of `compound module`.

```
package tokenring;
module Couple
{
    gates:
        input  min;
        output mout;
    submodules:
        n1: TRNode;
        n2: TRNode;
    connections:
        min-->n1.in;
        n1.out-->n2.in;
        n2.out-->mout;
}
```

Listing 6.18. *Definition of a compound module*

6.10.4. *The system*

We define two topologies (see Listing 6.19):

1) the first (`ThreeNodes`) is formed by three `TRNode` entities;

2) the second (`TwoCouplesOneNode`) is made up of two `Couple` (`compound module`) and one `TRNode` (`simple module`).

```
package tokenring.simulations;
import tokenring.TRNode;
import tokenring.Couple;
network ThreeNodes
{
    submodules:
        n1: TRNode;
        n2: TRNode;
        n3: TRNode;
    connections:
        n1.out --> n2.in;
        n2.out --> n3.in;
        n3.out --> n1.in;
}
network TwoCouplesOneNode
{
  submodules:
    m1: Couple;
    m2: Couple;
    n : TRNode;
  connections:
    m1.mout --> n.in;
    n.out --> m2.min;
    m2.mout --> m1.min;
}
```

Listing 6.19. *Definition of two models*

6.10.5. *Configuration of the simulation and its scenarios*

Simulations are conducted on both models (see Listing 6.20), through the configuration of, respectively, `TR3N` (with the model `ThreeNodes`) and the configuration `TR2C1N` (with the model `TwoCouplesOneNode`).

It is important that the identifiers are kept unique through all the network. We also show through these two examples various ways to set parameters, i.e. to create different simulation scenarios. In `TR3N` for example, `n1` is provided with a higher activity rate, `n2` with a higher loss rate and `n3` with a longer activity duration.

```
[General]
sim-time-limit = 2000s
#debug-on-errors = true
[Config TR3N]
network = ThreeNodes
# n3 has  the highest priority
**.n1.NodeId=1
**.n2.NodeId=2
**.n3.NodeId=3
**.n1.TokenHoldIndicator = bernoulli(0.5)
**.n2.LossIndicator = bernoulli(0.4)
**.n3.ActionDuration = uniform(4s, 8s)
[Config TR2C1N]
network = TwoCouplesOneNode
# networkwide setting:
**.m1.n1.NodeId=1
**.m1.n2.NodeId=2
**.m2.n1.NodeId=3
**.m2.n2.NodeId=4
**.n.NodeId=5
# concerns EVERY node
**.TokenPassingDelay = 0.02s
# concerns the two n1 nodes of both m1 et m2
**.n1.TokenHoldIndicator = bernoulli(0.5)
# concern the whole m2
**.m2.*.CheckTokenInterval = 30s
# individual setting
**.m1.n1.ActionDuration = uniform(4s, 8s)
**.m2.n2.LossIndicator = bernoulli(0.4)
```

Listing 6.20. *Configuration of simulation*

We notice once again that all the parameters are configurable, both numerically and by means of the probability distribution.

We could thus test the behavior of the algorithm under various situations (token loss ratio, durations of activities, delay of token-passing, etc.). For a

real study, probes will have of course to be set up, according to the aim of the study. We should also think about the quality of random variates by assigning various substreams of PRNG to different r.v. that have to be generated, in order to minimize the risk of an artificial dependence. We recall that these points are common practices to which we should pay attention when conducting simulations. For the sake of simplicity, we did not put any of these in this example, because our focus here is the coding of the model, and also because these points have already been addressed previously.

6.11. Building large systems: an example with `INET`

The example of section 6.10 provided a model which is built completely from the scratch. Most of the time, we will not carry out an *ex nihilo* modeling of an entire system, but rather have to *integrate* a new or modified component into a larger system. We provide hereafter such an example, with the `INET` package which provides a rather comprehensive modeling of various components of the Internet.

As already stated, our goal is neither the presentation of `OMNeT++` nor `INET`, but rather the illustration of some common modeling practices through them. So, we will use those functionalities as less as possible which are specific to `INET` and/or `OMNeT++`. The following example is actually a very *simplified adaptation* from the `lans` examples among those of `ethernet` given in `INET V2.3`, under `OMNeT++ V4.4.1`.

6.11.1. *The system*

As it is about adding a new component, we will adopt a *top-down* style presentation. The target system can roughly be interpreted as three applications communicating through Ethernet. We create a `project` under the name of `myethernet` and the homonym *namespace*. This is specified in the `package.ned` file under the `src` subdirectory (see Listing 6.21)

```
package myethernet;
@namespace(myethernet);
```

Listing 6.21. *The `myethernet` project and its namespace in* `package.ned`

For the sake of simplicity, the communication stack is limited to the LLC/MAC[5] layer. The communication entity is entirely specifically coded (see section 6.11.3), whereas the basic networking components (LCC, MAC, Ethernet Bus) are taken or composed (see section 6.11.2) of pre-built modules of the INET package.

The model of the system is called MyBusLAN and is defined in MyEthernet.ned under simulations subdirectory. (see Listing 6.22 with comments). It is formed by three entities, namely center, user1 and user2, of the same nature (module MyApp). They are connected together through a LLC-capable Ethernet card (module MyEthernetLLC) via a bus (EtherBus) with UTP Cat.5 type cables (MyUTPCat5).

It is to be pointed out that the model is put as simple as possible. For this reason, a Bus is used, even though it is no longer the standard way to interconnect stations. If we had chosen to use a *switch*, we would spend quite a long part to present and discuss about the *configuration* of switch under INET, which is actually specific to INET.

```
package myethernet.simulations;
// My own modules
import myethernet.MyApp;
// Modules composed from basic Inet
import myethernet.MyEthernetLLC;
// OMNET-INET predefined models
import ned.DatarateChannel;
import inet.linklayer.ethernet.EtherBus;
network MyBusLAN
{
    types:
        channel MyUTPCat5 extends DatarateChannel
        {
            delay = 0.1us;
            datarate = 100Mbps;
        }
    submodules:
        // Applis
```

5 LLC stands for *Logical Link Control* which is an upper (sub-)layer of MAC (*Medium Access Control*), the lower (sub-)layer. MAC controls transmission through physical media such as Ethernet. The couple LLC/MAC constitute jointly the *Link Layer* by providing a *logical* point-to-point communication channel for each communication peer.

```
        center: MyApp;
        user1:  MyApp;
        user2:  MyApp;
        // Ethernet Cards
        llc_c:  MyEthernetLLC;
        llc_u1: MyEthernetLLC;
        llc_u2: MyEthernetLLC;
        // Medium
        bus: EtherBus {
            parameters:
                positions = "10 20 30";
                                // 1us=200m
                propagationSpeed = 2e8 mps;
            gates:
                ethg[3];
        }
    connections:
        // Physical connections
        bus.ethg[0] <--> MyUTPCat5 <--> llc_c.ethg;
        bus.ethg[1] <--> MyUTPCat5 <--> llc_u1.ethg;
        bus.ethg[2] <--> MyUTPCat5 <--> llc_u2.ethg;
        // Application/Card interfacing
        center.out --> llc_c.FromUpper;
        llc_c.ToUpper --> center.in;
        user1.out --> llc_u1.FromUpper;
        llc_u1.ToUpper --> user1.in;
        user2.out --> llc_u2.FromUpper;
        llc_u2.ToUpper --> user2.in;
}
```

Listing 6.22. *An Ethernet-based communication system (*`MyEthernet.ned`*)*

6.11.2. *Ethernet card with LLC*

The module `MyEthernetLLC` (see Listing 6.23) is a `compound module`. It is created from existing basic protocol entities, respectively the `EtherLLC` and `IEtherMAC` modules, both in the `linklayer` rubric of the `INET` package, that we `import`. This is an example of the modular construction power of simulation tools based on OOP language (here `C++`) and assisted by an auxiliary configuration and linking language (here `NED`).

We want to point out that the LLC layer assures the interface with higher layer through the Service Access Point (*SAP*) mechanism. Roughly speaking, a *SAP* is an identifier for a particular protocol entity. The way the association is set will be detailed in section 6.11.3.2.

```
package myethernet;
import inet.linklayer.IEtherMAC;
import inet.linklayer.ethernet.EtherLLC;
module MyEthernetLLC
{
    parameters:
        // by default use CSMA/CD
        bool csmacdSupport = default(true);
        // ~EtherMAC or ~EtherMACFullDuplex
        string macType = default(csmacdSupport ?
                "EtherMAC" : "EtherMACFullDuplex");
    gates:
        inout ethg;
        input FromUpper;
        output ToUpper;
    submodules:
        llc: EtherLLC {
            gates:
                upperLayerIn[1];
                upperLayerOut[1];
        }
        mac: <macType> like IEtherMAC;
    connections:
        // Interface with Application
        FromUpper --> llc.upperLayerIn[0];
        llc.upperLayerOut[0] --> ToUpper;
        // Internal connections of prebuilt modules
        mac.upperLayerOut --> llc.lowerLayerIn;
        llc.lowerLayerOut --> mac.upperLayerIn;
        // Interface with Physical connection
        mac.phys <--> ethg;
}
```

Listing 6.23. *An Ethernet card with LLC (*`MyEthernetLLC.ned`*)*

6.11.3. *The new entity* MyApp

We are now going to present the new module that we truly code and add to the existing INET environment. For a real project, this new part can be a rather complex subsystem. Here, for the sake of simplicity, we take the minimal form, i.e. we take again our first example, i.e. a simple sender (MySource, see section 6.7.1.1) and a simple receiver (MySink, see section 6.7.1.2). Here, we put them together in a same entity. The latter acts as a *sender* when it receives a *self-message*, which means that *it is time* to send a packet; and it acts as a *receiver* otherwise, i.e. upon arrival of a *real* message modeling a data packet. The entity, named MyApp, is described in MyApp.ned (see Listing 6.24) and implemented in MyApp.cc (see Listing 6.25), both under the src subdirectory as usual.

6.11.3.1. *Description of* MyApp

We first give some comments on the description of MyApp:

– each communication entity must know how to reach its destination(s). Classical mechanisms in the Internet are ARP for matching between an IP address and a MAC address, or the DNS for matching between the literal name of an entity and its IP address. Here, we need to know the MAC address and the LLC-level SAP. We have chosen here the simplest way, from the view point of coding, which consists of *manually* setting these address. For the sake of coding simplicity, we also choose to consider a *fixed* three-peers scenario by *hard-coding* the two potential destinations, referred symbolically as *Sink0* and *Sink1*. We prefer this *manual* setting and coding, instead of using more generic and automatic methods that INET provides, basically due to the goal of this example, in order to avoid long presentation of these INET-specific methods;

– the traffic is parametric, for arrival process (through sendIaTime) as well as for packet length (through Length). In addition, the traffic dispatching is modeled by a binary indicator Sink0vsSink1.

```
package myethernet;
simple MyApp
{
    parameters:
        // Sink0, Sink1: the 2 dest App.
        string Sink0Adrs; // Mac adrs
        string Sink1Adrs;
        int    Sink0Sap;  // Their Sap
```

```
        int    Sink1Sap;
        int    ThisAppSap; //
        // switch indicator: Bernoulli
        volatile int Sink0vsSink1;
        // Trafic (arrival, volume) parameters
        volatile double sendIaTime @unit(s);
        volatile int    Length; // in byte
    gates:
        input  in;
        output out;
}
```

Listing 6.24. *Description of* `MyApp` *(*`MyApp.ned`*)*

6.11.3.2. *Implementation of* `MyApp`

The implementation of `MyApp` deserves longer comments:

– the `initialize()` procedure is slightly different from all the previous models. Here, it is done in two stages, namely the stage 0 and the stage 3. In a large system, it is actually practical to be able to split initialization of various and inter-dependent modules into several *waves*. `OMNeT++` provides this functionality. Here, it is about a simple coding illustration;

– the setting of the `SAP` of an application is done here through the means of `registration` which is used by the `EtherLLC` module. It consists of sending a packet to the LLC module with registration indication and the entity's SAP as `DSAP` (Destination SAP). This is a pure programming choice by `INET`, each simulation language has its (numerous) *rules* that we have to learn. This setting needs to be done only once (at the very beginning via the indicator `isDSapSet`);

– the data packet must conform to the Ethernet format and with *Data* indication (`IEEE802CTRL_DATA`). The MAC and LLC headers (at least destination MAC address and DSAP) are set via the format `Ieee802Ctrl`. These syntaxes are also part of programming *rules* of the `INET` package;

– we have put only one *probe*, which measures the end-to-end delay, and thus provides some basic statistics such as the count of received packets.

```
#include <omnetpp.h>
#include "EtherAppCli.h"
#include "EtherApp_m.h"
#include "Ieee802Ctrl_m.h"
```

```
namespace myethernet{
class MyApp : public cSimpleModule
{
  protected:
//Info (Mac/Sap) of the 2 destinations
    MACAddress  DA_Sink0;
    MACAddress  DA_Sink1;
    int         SAP_Sink0;
    int         SAP_Sink1;
    //
    int ThisAppSap;
    // For the DSap set-up purpose
    bool        isDSapSet;
// stat. on received packet number and e2e delay
    cStdDev     myStat;
    //
    virtual void initialize(int stage);
    virtual int numInitStages() const { return 4; }
    virtual void handleMessage(cMessage *msg_in);
    virtual void finish();
};
Define_Module(MyApp);
void MyApp::initialize(int stage)
{
    cSimpleModule::initialize(stage);
    if (stage == 0) {
    isDSapSet=false;
    DA_Sink0.setAddress( par("Sink0Adrs").
      stringValue() );
    DA_Sink1.setAddress( par("Sink1Adrs").
      stringValue() );
    SAP_Sink0=par("Sink0Sap").longValue();
    SAP_Sink1=par("Sink1Sap").longValue();
    ThisAppSap=par("ThisAppSap").longValue();
    EV << "Init Saps " << ThisAppSap << " ";
    EV <<  SAP_Sink0 << " " << SAP_Sink1 << endl;
    } else if (stage == 3) {
    // every module is now initialized
    // start the sending process
    scheduleAt(simTime(), new cMessage);
    EV << " init Done  stage  " << stage << endl;
    }
} // end init
```

```
void MyApp::finish()
{
    recordScalar(" End-to-End delay", simTime());
    myStat.record();
    EV << " Count: " << myStat.getCount();
    EV << " Mean D(e2e): " << myStat.getMean() << endl;
}
void MyApp::handleMessage(cMessage *msg_in)
{
    if (isDSapSet==false) {
        isDSapSet=true;
        EV << "registering DSAP" << ThisAppSap << "\n";
        Ieee802Ctrl *etherctrl = new Ieee802Ctrl();
        etherctrl->setDsap(ThisAppSap);
        cMessage *msg = new cMessage("register_DSAP",
                          IEEE802CTRL_REGISTER_DSAP);
        msg->setControlInfo(etherctrl);
        send(msg, "out");
    }
    if ( msg_in->isSelfMessage() ) {
    // send a new Ethernet packet
    EtherAppReq *pkt = new EtherAppReq("data",
       IEEE802CTRL_DATA);
    pkt->setByteLength(par("Length").longValue());
    // set adequately Ethernet MAC/LLC Header
    Ieee802Ctrl *etherctrl = new Ieee802Ctrl();
    etherctrl->setSsap(ThisAppSap);
    // Dispatch
    if ( par("Sink0vsSink1").longValue() > 0 ) {
        etherctrl->setDest(DA_Sink1);
        etherctrl->setDsap(SAP_Sink1);
    }
    else {
        etherctrl->setDest(DA_Sink0);
        etherctrl->setDsap(SAP_Sink0);
    }
    pkt->setControlInfo(etherctrl);
    //EV << "-OUT->>--  Send Packet " <<
etherctrl << endl;
    send(pkt, "out");
    // program the next send event,
    scheduleAt(simTime()+par("sendIaTime").doubleValue(),
       msg_in);
```

```
    } // self msg
    else { // Data in: Sink behavior
      simtime_t d=simTime() - msg_in->getCreationTime();
      myStat.collect(d);
      // EV << ">>--IN-- Receive Packet " << endl;
      delete msg_in;
    }
} // end handleMessage
}// namespace: myethernet
```

Listing 6.25. *Implementation of* `MyApp` *(*`MyApp.cc`*)*

6.11.4. *Simulation*

Sample simulations are run through the usual configuration file `omnetpp.ini` (see Listing 6.26) under the `simulations` subdirectory. Here are some comments.

– we have adopted a *minimalist* configuration, which is reduced to the necessary settings: various options and functionalities at simulation configuration stage having been addressed through previous examples. Despite this reduction, there are still quite a number of parameters to be set. Thus, a careful check, and, for parametrized simulation campaigns, some automatized configuration generator, are definitely recommended;

– the simulation scenario is the follows:

- the entity `center` sends a Poisson flow with $\lambda_c = 1$ packet/s, and dispatches 30% of them to `user1` (*sink0*) and 70% of them to `user2` (*sink1*);

- the entity `user1` sends a Poisson flow with $\lambda_1 = 0.1$ packet/s, and dispatches 10% of them to `center` (*sink0*) and 90% of them to `user2` (*sink1*);

- the entity `user2` sends a Poisson flow with $\lambda_2 = 0.2$ packet/s, and dispatches them evenly to `center` (*sink0*) and `user1` (*sink1*);

- all the packets have the same length distribution (uniform over $[46, 1000]$).

– We give observations from a $10,000$ s trial run, under the form "simulation data (theoretical prediction)": `center` receives 1050 (1000) packets, `user1` : 4037 (4000) and `user2` : 8015 (7900) for a total of 13102 (13000) packets.

```
[General]
sim-time-limit = 10000s
[Config MyBusLAN]
network = MyBusLAN
**.llc_c.mac.address="01:02:03:04:05:05"
**.llc_u1.mac.address="01:02:03:04:05:06"
**.llc_u2.mac.address="01:02:03:04:05:07"
# common packet length distribution
**.Length=intuniform(46, 1000)
# center: 05:05
**.center.Sink0Adrs = "01:02:03:04:05:06"
**.center.Sink1Adrs = "01:02:03:04:05:07"
**.center.ThisAppSap = 10
**.center.Sink0Sap =11
**.center.Sink1Sap =12
**.center.sendIaTime = exponential(1s)
**.center.Sink0vsSink1 = bernoulli(0.7)
# user1 : 05:06
**.user1.Sink0Adrs = "01:02:03:04:05:05"
**.user1.Sink1Adrs = "01:02:03:04:05:07"
**.user1.ThisAppSap = 11
**.user1.Sink0Sap =10
**.user1.Sink1Sap =12
**.user1.sendIaTime = exponential(10s)
**.user1.Sink0vsSink1 = bernoulli(0.9)
# user2 : 05:07
**.user2.Sink0Adrs = "01:02:03:04:05:05"
**.user2.Sink1Adrs = "01:02:03:04:05:06"
**.user2.ThisAppSap = 12
**.user2.Sink0Sap =10
**.user2.Sink1Sap =11
**.user2.sendIaTime = exponential(5s)
**.user2.Sink0vsSink1 = bernoulli(0.5)
```

Listing 6.26. *MyEthernet: simulation configuration (*`omnetpp.ini`*)*

6.11.5. *Conclusion*

This case study provides a very simple example of how to integrate customized modules into a larger system. Through this example, several points are illustrated:

– the power and practice of modular construction;

– the importance of mastering the underlying technology behind a pre-built model, in order to understand the modeling approach of the latter;

– the effort to commit in order to identify and understand particular rules, constraints and syntax which are used by a simulation language for their pre-built models;

– the need of well organized, preferably systematically, simulation scenarios, which could be rather complex for large systems.

We would also like to highlight at last, and far from the least, a point concerning the data collection. In this example, just one probe is *explicitly* set. The latter provides the number of packets that each terminal received. In fact, it is not useful to set this specific probe. This value is already provided by a built-in signal of LLC, namely the `passedUpPk`, through its `count` field, the same signal can provide the total received bytes as well. Actually, pre-built LLC and MAC modules have quite a number of pre-defined `signals`, giving statistics on various quantities and phenomena, for instance, collision, received data in bytes, number of bad packets, etc. These pre-defined probes are available and can be explored for various purposes. This is a general observation for all pre-built modules.

PART 2

Queueing Theory

7

Introduction to the Queueing Theory

In this chapter, we introduce the basic concepts of the queueing theory. This tool is largely used to study, in an analytical manner, the behavior of any system receiving randomly submitted requests that have to be processed with the consumption of some resources, and some of the requests can be kept on standby in a queue prior to their processing.

7.1. Presentation

Let us take the example of the transmission of an Internet Protocol (IP) packet through a networking interface, in order to highlight various types of resources that this packet will consume, as well as some performance indicators:

– if the packet is the only one (a frequent phenomenon in case of weak traffic), it will be transmitted at once, if not, it must be put on standby (frequent phenomenon in case of sustained traffic), thus requiring some storage capacity;

– it is always necessary to get some storage capacity (buffer memory) to keep the waiting packets as well as the newly arrived packets during the transmission of the current one;

– the transmission of IP packet consumes a resource, in our example, it is the bandwidth of the networking interface;

– one could be interested by the following questions:

- what is the number of waiting packets in the buffer?

- in which manner the packets are submitted?

Performance Evaluation by Simulation and Analysis with Applications to Computer Networks, First Edition. Ken Chen. © ISTE Ltd 2015. Published by ISTE Ltd and John Wiley & Sons, Inc.

- what is the distribution of the size of the packets (which directly determines the bandwidth consumption and the storage that a packet claims)?

- is the offered bandwidth sufficient to support the submitted traffic?

- what is the mean waiting time for a given level of traffic?

- what should be the size of the buffer so that the loss ratio, due to rejection of packets in case of overflow of the buffer, is lower than a given threshold?

This system formed by a networking interface (bandwidth) and its associated buffer (random access memory (RAM)) can be modeled by a queue. Figure 7.1 gives the usual presentation of a queue, where the packets of our example are called in a generic way *customers*, and the processing mechanism (transmission in our example) is called *server*. It is clear that the *service* here is the transmission of packet, at the price of the consumption of a fraction of the bandwidth and, if necessary, after having occupied some place in the buffer.

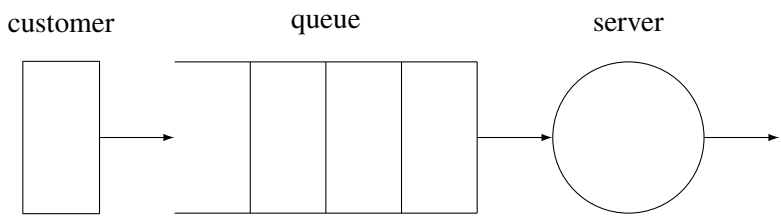

Figure 7.1. *A queueing system*

We could also consider a more complex system of which the previous queue is the first stage: packets are directed toward another router who dispatches them toward various directions. In a similar manner, we can set a queueing model for the second router. We have here a configuration with two queues put in series. Continuing our analysis by imaging a router behind each direction of the second router, we obtain a network of queues.

The queueing theory aims to study the stochastic systems in which a service mechanism, called *server*, takes a certain time to process objects called *customers*. The customers arrive in a random way and claim various service times. If the server cannot manage to process entirely the current

customer before arrival of the subsequent customers, the latter has to be kept in a waiting queue. A buffer (storage resource) is required for keeping temporary the waiting customers. A queueing model can be applied to situations where there is submission of requests claiming some processing which consumes some resources. The resources consumption does not explicitly appear in the model, but only implicitly through the service duration.

There are many works devoted to the queueing theory. A classical one is that of Kleinrock [KLE 75], which is one of the references on the matter, as well as the monumental work of Cohen [COH 69]. Among the very rich literature, the author would like to cite a few of them [COO 81, TAK 91, GEL 98, ROB 03, BRE 05, DAI 05, BOL 06, VAN 06, BHA 08, DAT 08, DEC 12]. The book [VAN 06] also presents graph-theory-based modeling. In [YUE 09], one can find some sample applications of analytical modeling to computer networks.

7.2. Modeling of the computer networks

The queueing theory is a very useful tool for the modeling and the performance evaluation of computer networks. Indeed, a computer network consists of a sequence of hardware and software components through which the messages are convoyed from a point to another within the network.

EXAMPLE 7.1. (IP packets crossing a router) – Several resources are implied and various models can be established according to the goal of study. A simple model consists of modeling the router by only one queue, where the server represents the routing process and the buffer represents the storage capacity of the router.

7.3. Description of a queue

The principal elements that characterize a queue are as follows:

– the arrival distribution (in which manner the requests are submitted);

– the service mechanism that is characterized by:

- the service duration's distribution;

- the number of servers;

- the service discipline (how the customers are scheduled);

- the storage capacity of the system.

To define in a concise way a queue, we generally use the *notation of Kendall* who takes the following form:

$$A/B/C/K/Z/ \qquad [7.1]$$

where

– A and B are, respectively, the distribution of arrival and service;

– C gives the number of *identical* servers;

– K: maximum capacity (including the customer(s) under service);

– Z: service discipline.

Hereafter, we give notations about the distributions of arrival and services that we will use in the following:

– G: General distribution;

– GI: General distribution with independent and identically distributed (i.i.d.) random variables (r.v.);

– M: Exponential distribution (Markov distribution);

– D: Constant distribution (deterministic).

Among the most used disciplines of service, we can quote:

– FIFO: *first in first out*;

– LIFO: *last in first out*;

– SPF: *shortest processing time first*;

– HOL: *head of the line* (queue with static and strict priorities);

– EDF: *earliest deadline first*.

EXAMPLE 7.2.– $M/GI/1/10/FIFO$ defines a queue having a single server with a buffer which can keep up to nine customers on standby. The customers are arranged according to their arrival order (FIFO discipline). The arrival process has the Poisson distribution (Markov), the service times are distributed according to a general distribution and are mutually independent.

By default, we suppose that:

– the queue has an infinite buffer if K is omitted ($K = \infty$);

– the customers are served in FIFO order if Z is omitted (Z = FIFO).

In the remainder, we will mainly present the $M/M/.$ queues, as well as $M/GI/1$ queues and two of their variants (queues with server vacation, HOL queues).

7.4. Main parameters

Let us consider a queue and focus on a given customer, say the ith customer (denoted as C_i) since the very beginning. We will follow its evolution (or *trajectory*) (see Figure 7.2) in the queue. The main parameters of interest of this customer are:

– the moment of its arrival τ_i;

– the moment when its service starts σ_i;

– the duration of its waiting time W_i ($W_i = \sigma_i - \tau_i$);

– the moment of its departure δ_i;

– duration of its service S_i ($S_i = \delta_i - \sigma_i$);

– duration of its sojourn in the queue T_i ($T_i = \delta_i - \tau_i = W_i + S_i$).

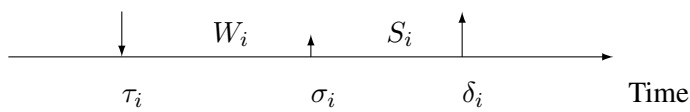

Figure 7.2. *Trajectory of a customer*

Let us now consider the flow of all customers. We are interested by the following parameters:

– $\Delta_i = \tau_{i+1} - \tau_i$: the interarrival interval between the neighbor customers C_i and C_{i+1};

– $A(t)$: the cumulative number of arrivals from the beginning (0) till the instant t;

– $B(t)$: the cumulative number of departures from the beginning (0) till the instant t;

– $N(t)$: the number of customers in the queue at the instant t ($N(t) = A(t) - B(t)$).

7.5. Performance indicators

7.5.1. *Usual parameters*

The performance of a system modeled by a queue can be evaluated through many parameters and according to various points of view. Here are some examples of the usual parameters:

– sojourn time of a customer, it can also be interpreted as the response time of the system;

– number of customers in the queue (seen by a new customer on its arrival);

– probability of finding the queue empty (immediate service);

– utilization ratio of the server;

– if we suppose that the queue is of finite capacity (a buffer with limited capacity), the probability of finding the queue full becomes also an eligible indicator. This parameter can also be seen as the loss (or rejection) ratio.

7.5.2. *Performance in steady state*

The parameters which we have just presented are defined specifically at a particular moment t and/or for a particular customer C_i. The study on these parameters is thus valid only for that moment and/or that customer, which offers few interest.

We are generally interested by a system in *steady state*. It is the situation where the indicators (most of them are mean values) can be considered as *timely invariant*. Let us take, for example, the average number of customers in the queue, denoted as $E[N]$. Formally, $E[N]$ can be calculated as:

$$E[N] = \lim_{t \to \infty} \frac{1}{t} \int_0^t N(u) du.$$

This limit does not exist for all the queues. In particular, if the server is overloaded, the queue's length ($N(u)$) will increase indefinitely with time. When it exists, which is the case for the majority of the systems whose study is of real interest, it is said that the system is in *steady state* (or *equilibrium state*). In a similar way, we also define:

– the arrival rate λ in steady state:

$$\lambda = \lim_{t\to\infty} \frac{A(t)}{t}$$

– the departure rate μ in steady state:

$$\mu = \lim_{t\to\infty} \frac{B(t)}{t}$$

– the mean sojourn time $E[T]$ in steady state:

$$E[T] = \lim_{t\to\infty} \frac{\sum_{i=1}^{A(t)} T_i}{A(t)}$$

7.6. The Little's law

7.6.1. *Presentation*

The *Little's law* provides, through a very simple formula and for a system in its very generic form (a black box), a relationship between:

– $E[N]$: the average number of customers in the system;

– $E[T]$: the average sojourn time in the system and

– λ: the arrival rate (i.e. mean arrival number per time unit) of the customers admitted into the system.

Indeed, in 1961, J. Little found the following formula:

$$E[N] = \lambda E[T] \qquad [7.2]$$

The proof of this formula can be done in an almost intuitive manner. Indeed, the number $N(t)$ is the difference between the arrivals and the departures. Let t be an instant at which the system becomes (again) empty, i.e. $N(t) = 0$. One

can note graphically, by drawing $N(t)$ (which is a step function) as a function of t, that:

$$\int_0^t N(u)du = \sum_{i=1}^{A(t)} T_i$$

which becomes:

$$\frac{1}{t}\int_0^t N(u)du = \frac{A(t)}{t}\frac{\sum_{i=1}^{A(t)} T_i}{A(t)}. \qquad [7.3]$$

Note that the formula [7.3] is only valid for t such that $N(t) = 0$. We assume now that the steady state exists. This means that there are an infinity of moments t such as $N(t) = 0$. We can then make $t \to \infty$ in equation [7.3] which leads to the Little's law given by equation [7.2].

The capital role of the assumption of steady state, i.e. the situation $N(t) = 0$ comes back forever surely (i.e. with a non-null probability), should be underlined. We will see that it is always this condition that one seeks to establish to put the system in steady state.

7.6.2. *Applications*

The Little's law establishes a very important relationship among three fundamental parameters of a system, without particular assumption except that of the existence of the average values (i.e. existence of the steady state). These three parameters are related, respectively, to the intensity of the requests (λ), to the need for (temporary) storage ($E[N]$), as well as to the processing capacities which determine $E[D]$, the sojourn time (or the response time of the system). The following example gives an application of the Little's law.

EXAMPLE 7.3.– We consider a router. We note that it receives on average 3,000 packets per second. It is supposed that the packets have the same length: 1,000 bytes. It is also supposed that the flow of packets can be modeled as a Poisson process. The router has a limited buffer. In case where the buffer is full, newly arriving packets are rejected. We also observe that the packets which are admitted into the router spend on average 100 ms in it:

1) For each one of the configurations of the buffer, indicate, with justification, if it is: Sure (S), Possible (P), Rare (R) and Impossible (I) to have rejections of packets:

i) Buffer size: $B = 20$K bytes;

ii) Buffer size: $B = 512$K bytes;

iii) Buffer size: $B = 4$M bytes;

iv) Buffer size: $B = 1$G bytes.

For the sake of simplicity, K (idem for M, G) is interpreted in its mathematical definition $(1,000)$ and not by its value in computer world $(1,024)$.

2) We assume from now on that the packets are submitted according to a Pareto distribution, with an average of 2,000 packets per second. It is also supposed that the length of the packets is no longer constant but is random with an average of 1,000 bytes. The average sojourn time of admitted packets remains the same (100 ms). Answer the same questions as previously.

Solution:

1) This is a direct application of the Little's law. Here, $\lambda = 3,000$ packets/sec, and $E[D] = 0.1$ sec. Thus, N_a, the average number of the packets remaining in the router, by supposing that all the packets are admitted, is $N_a = \lambda\, E[D] = 300$ packets. This corresponds to a storage capacity of $C_a = 1,000\, N_a = 300$K bytes. We note $N(t)$ the actual number of packets in the router at the moment t, whose maximum value will be denoted as N_m. It is obvious that $E[N] < N_m$, and generally $N(t)$ is random and fluctuates around $E[N]$. The more irregular the arrivals are, the larger the difference between $E[N]$ and N_m will be, as well as the fluctuation of $N(t)$:

i) Sure: indeed, $B < C_a$, $N_m = \frac{20K}{1,000} = 20 < N_a$, a fortiori, $E[N]$ cannot exceed N_m, some packets will surely be rejected;

ii) Possible: indeed, $B > C_a$, $N_m = \frac{512K}{1,000} = 512 > N_a$, it is thus possible that all the packets being admitted, if $N(t)$ does not fluctuate too much;

iii) Rare: indeed, $B \gg C_a$, $N_m = \frac{4M}{1,000} = 4K \gg N_a$, it is rare that $N(t)$ could reach N_m. However, the risk always exists;

iv) Rare: indeed, in this case, the difference between B and C_a, or, equivalently, between N_a and N_m, is even more significant than the case with $B = 4$M bytes, it is even less probable that $N(t)$ could reach N_m. However, since we are in a stochastic context, the risk exists always theoretically.

2) The Little's law does not depend on the nature of the probability distributions. We can thus apply the preceding reasoning. Here, $E[N] = 200$, by remaking the same comparisons, we get the same conclusions.

8

Poisson Process

In this chapter, we present the Poisson process which is the most used arrival process in analytical modeling. It is actually a process which offers a faithful model for many phenomena.

8.1. Definition

We present below two definitions of the *Poisson process*, the first definition provides a more intuitive physical interpretation, whereas the second definition is better suited for calculations.

8.1.1. *Definition*

DEFINITION 8.1.– *A Poisson process with rate* λ *is a counting process* $\{N(t), t \geq 0\}$*,* $N(0) = 0$*, verifying*:

1) *ordered arrivals*:

$$P[N(t) = 0] = 1 - \lambda t + o(t)$$

$$P[N(t) = 1] = \lambda t + o(t)$$

where $o(t)$ *is a function of* t *verifying* $\lim_{t \to 0} \frac{o(t)}{t} = 0$;

2) *stationary increments*:

$$P[\,[N(t+h) - N(t)] = n\,] = P[N(h) = n], \;\; h > 0, \; t > 0;$$

Performance Evaluation by Simulation and Analysis with Applications to Computer Networks, First Edition. Ken Chen. © ISTE Ltd 2015. Published by ISTE Ltd and John Wiley & Sons, Inc.

3) *independent increments*:

$$\begin{array}{l} P[\{\,[N(t+h)-N(t)]=n_1\}\,\&\{\,N(t)=n_2\}] \\ =P[\,[N(t+h)-N(t)]=n_1\,]\,P[N(t)=n_2] \end{array}.$$

Here are some comments and observations about this definition:

– the first condition implies $P[N(t) \geq 2] = o(t)$. Actually,

$$P[N(t) \geq 2] = 1 - (P[N(t) = 1] + P[N(t) = 0]) = o(t)$$

This means that the probability of having several arrivals over one infinitesimal period is negligible. This justifies the term "ordered arrivals", i.e. there could be only one single arrival at any given instant and there are no grouped arrivals. In other words, the arrivals are "serialized" in time;

– the second condition implies that the increment does not depend on the instant t of counting, but solely on the duration of the counting period (h);

– the third condition implies that the past (the cumulative arrivals till t) is independent of the future evolution: $N(t+h) - N(t)$ is the number of new arrivals on $(t, t+h)$.

8.1.2. *Distribution of a Poisson process*

It can be shown that Definition 8.1 is equivalent to the following one, which has the advantage of being much suitable for calculations.

DEFINITION 8.2.– *A Poisson process with rate λ is a counting process $\{N(t), t \geq 0\}$, $N(0) = 0$. The increments are independent, and the number of the arrivals during an interval of duration t has the following distribution*:

$$P[N(t) = n] = e^{-\lambda t}\frac{(\lambda t)^n}{n!},\ n \in \mathbb{N}, \lambda > 0, t \geq 0 \qquad [8.1]$$

In the remaining, we will use Definition 8.2 rather than 8.1, for its better mathematical tractability. Hereafter is a first example with the computation of $E[N(t)]$.

For a Poisson process $\{N(t), t \geq 0\}$ with rate λ, we have: $\forall t > 0$, $E[N(t)] = \lambda t$. Indeed:

$$E[N(t)] = \sum_{n=0}^{\infty} n\, e^{-\lambda t} \frac{(\lambda t)^n}{n!} = e^{-\lambda t}(\lambda t) \sum_{n=1}^{\infty} \frac{(\lambda t)^{n-1}}{(n-1)!}$$
$$= e^{-\lambda t}(\lambda t) e^{\lambda t} = \lambda t$$

With $t = 1$, we have $E[N(1)] = \lambda$. Thus, we get the physical interpretation of the parameter λ, which is precisely the average number of the arrivals during a time unit ($t = 1$), This justifies the term *arrival rate*. Concerning the variance, by noticing that

$$Var[X] = E[X^2] - (E[X])^2, \text{ and } E[X^2] = E[X(X-1)] + E[X]$$

we get in a similar manner:

$$Var[N(t)] = \lambda t = E[N(t)] \qquad [8.2]$$

8.2. Interarrival interval

8.2.1. *Definition*

Consider a Poisson process $\{N(t), t \geq 0\}$ with rate λ. Let τ_i be the arrival time of the i-th customer denoted by C_i. By convention, $\tau_0 = 0$. Let $\Delta_i = \tau_{i+1} - \tau_i$, Δ_i is the duration of the *interarrival* interval between the arrival of C_i and that of C_{i+1} (see Figure 8.1). It is clear that $\{\Delta_i\}$ forms a sequence of random variables (r.v.).

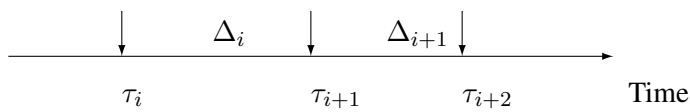

Figure 8.1. *Arrival instants and inter-arrival intervals*

8.2.2. *Distribution of the interarrival interval* Δ

We saw previously that the number of the arrivals between t and $t+h$, $N(t+h)-N(t)$, is independent of $N(t)$. In other words, the history of the arrivals until t has no impact on what will occur after this date. This *memoryless* property is an essential characteristic of the Poisson process (which belongs to the family of Markov processes). This property is expressed by:

$$P[N(\tau_i+h)-N(\tau_i)=0/N(x);x\leq\tau_i]=P[N(h)=0]=e^{-\lambda h}$$

The same property also implies that Δ_i is mutually independent.

However, by considering the arrival instants, we get:

$$\begin{aligned}&P[N(\tau_i+h)-N(\tau_i)=0/N(x);x\leq\tau_i]\\&=P[\tau_{i+1}-\tau_i>h/\tau_0,\ldots,\tau_i]\\&=P[\Delta_i>h/\tau_0,\ldots,\tau_i]=P[\Delta_i>h]\end{aligned}$$

Thus, $P[\Delta_i>h]=e^{-\lambda h}$, i.e. $P[\Delta_i\leq h]=1-e^{-\lambda h}$. We get straightforwardly:

THEOREM 8.1.– The $\{\Delta_i\}_{i\in\mathbb{N}}$ are i.i.d. r.v., having an exponential distribution with rate λ, i.e. for all $i\geq 0$ and all $t\geq 0$:

$$P[\Delta_i\leq t/\tau_0,\ldots,\tau_i]=P[\Delta\leq t]=1-e^{-\lambda t} \qquad [8.3]$$

This result has an interesting application in simulation by offering a way to generate a Poisson process. Indeed, a Poisson process with rate λ is obtained by generating a succession of instants, considered as arrival instants. These instants are separated by durations having the $Exp(\lambda)$ distribution (see section 3.6.3.1.1). Figure 8.2 gives a Poisson process generated in this way.

The mean and the variance of the interarrival interval distribution of a Poisson process are given by:

$$\begin{cases}E[\Delta]=\int_0^\infty t\lambda e^{-\lambda t}dt=\frac{1}{\lambda}\\Var[\Delta]=E[\Delta^2]-(E[\Delta])^2=\frac{1}{\lambda^2}\end{cases}$$

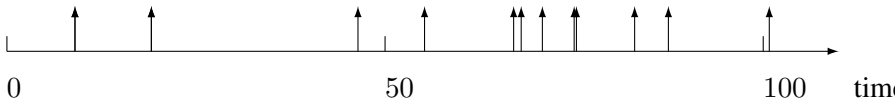

Figure 8.2. *Simulation of a Poisson process with rate* $\lambda = 0.1$

8.2.3. *Relation between* $N(t)$ *and* Δ

It was established that the reciprocal of the previous result (Theorem 8.1) is also true. Thus, we obtain yet another definition of a Poisson process.

THEOREM 8.2.– Let $\{\tau_i\}_{i\in\mathbb{N}}$ be the arrival instants of a counting process $N = \{N(t), t \geq 0\}$. N is a Poisson process with rate λ if and only if the interarrival intervals $\{\Delta_i = \tau_{i+1} - \tau_i\}_{i\in\mathbb{N}}$ are i.i.d. r.v. with an exponential distribution $Exp(\lambda)$.

This result gives us two ways to characterize and interpret a Poisson process: by the arrival numbers or by the arrival instants.

8.3. Erlang distribution

This is a distribution related to τ_n which is the arrival moment of the n-th customer. It is given by:

$$\forall n > 0, \forall t \geq 0, \;\; P[\tau_n \leq t] = 1 - \sum_{i=0}^{n-1} e^{-\lambda t} \frac{(\lambda t)^i}{i!} \qquad [8.4]$$

Indeed:

$$\begin{aligned} P[\tau_n \leq t] &= P[N(t) \geq n] = 1 - P[N(t) < n] \\ &= 1 - \sum_{i=0}^{n-1} P[N(t) = i] = 1 - \sum_{i=0}^{n-1} e^{-\lambda t} \frac{(\lambda t)^i}{i!} \end{aligned}$$

This distribution is known as the Erlang-n distribution, and is denoted by E_n.

8.4. Superposition of independent Poisson processes

DEFINITION 8.3.– *We call* superposition *of the processes* $N_i,\ i = 1, 2, \ldots, n$, *the process* N *defined by* $N(t) = \sum_{i=1}^{n} N_i(t)$.

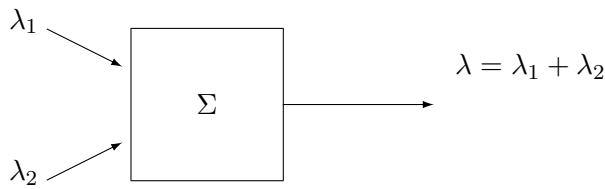

Figure 8.3. *Superposition of 2 independent Poisson processes*

Let us consider n Poisson processes $N_i = \{N_i(t), t \geq 0\}$ of rate λ_i, $i = 1, 2, \ldots, n$. Moreover, we assume that these processes are mutually independent. The following result applies to the superposed process.

THEOREM 8.3.– The superposition of n independent Poisson processes remains a Poisson process whose rate is equal to the sum of the rates of all these processes.

This result can be shown as follows. Indeed, it is obvious that the rate of the resulting process is the sum of the rates, due to flow conservation. However, it is less obvious that the resulting process is Poisson. We will show it by recurrence.

Let us start with two independent Poisson processes (see Figure 8.3), N_1 and N_2, with rates λ_1 and λ_2, respectively. Consider $N = N_1 + N_2$, due to the independence between N_1 and N_2 (third line of the following equation),

we have:

$$
\begin{aligned}
& P[N(t) = n] \\
&= \sum_{i=0}^{n} P[N_1(t) = i, N_2(t) = n - i] \\
&= \sum_{i=0}^{n} P[N_1(t) = i]\, P[N_2(t) = n - i] \\
&= \sum_{i=0}^{n} e^{-\lambda_1 t} \frac{(\lambda_1 t)^i}{i!} e^{-\lambda_2 t} \frac{(\lambda_2 t)^{n-i}}{n-i!} \\
&= e^{-(\lambda_1+\lambda_2)t} \frac{[(\lambda_1 + \lambda_2)t]^n}{n!} \sum_{i=0}^{n} \frac{n!}{i!(n-i)!} (\frac{\lambda_1}{\lambda_1 + \lambda_2})^i (\frac{\lambda_2}{\lambda_1 + \lambda_2})^{n-i} \\
&= e^{-(\lambda_1+\lambda_2)t} \frac{[(\lambda_1 + \lambda_2)t]^n}{n!}
\end{aligned}
$$

In conclusion, N is a Poisson process with rate $\lambda = \lambda_1 + \lambda_2$. The proposal is true for 2 flows. The remainder of the proof is obtained by recurrence with the same technique.

EXAMPLE 8.1.– The consolidated traffic sent out by the unique router of an Internet site is the sum of the local outgoing traffics. If these traffics are independent Poisson processes, then the overall outgoing traffic will also be a Poisson process.

8.5. Decomposition of a Poisson process

Let us consider the case of an Internet router, say R, which is connected to two other routers, say A and B. R has to dispatch the outgoing packets either toward A or toward B. Thus, there are two subtraffics toward A and B, respectively. The nature (datagram) of the Internet routing ensures that the dispatching operation is *independent* for each packet. Moreover, it is possible to get the proportion of the packets routed toward A. Or, in a dual manner, we may want to control this parameter in order to control the amount of each subtraffic. This proportion, say $p_1 = p$ for A (thus that for B is $p_2 = 1 - p$), is the probability that a packet is dispatched toward A. Assuming that the traffic

received by R is a Poisson process, we would be interested in the nature of these two subtraffics.

This problem can be modeled as follows. Let N denote the input process to R, N_1 (respectively, N_2) is the subtraffic of N which is dispatched to A (respectively, B). Obviously, $N(t) = N_1(t) + N_2(t)$. Upon each arrival in N, either N_1 is incremented by 1 and N_2 remains unchanged, or N_2 is incremented by 1 and N_1 remains unchanged. The probabilities of these two situations are, respectively, p_1 for the first and p_2 for the second. By definition, the processes N_1 and N_2 form a *decomposition* of the process N (see Figure 8.4).

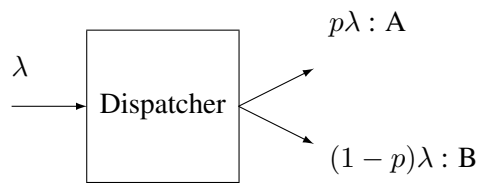

Figure 8.4. *Decomposition of a Poisson process*

THEOREM 8.4.– If N is a Poisson process with rate λ, then N_1 and N_2 are Poisson processes with rate $\lambda_i = p_i\lambda, i = 1, 2$. Moreover, N_1 and N_2 are independent.

This result can be shown as follows. Let us take $N_1(t)$. We have, while conditioning on all the possible values of $N(t)$:

$$\forall m \in \mathbb{N}, \quad P[N_1(t) = m] = \sum_{n=0}^{\infty} P[N_1(t) = m/N(t) = n]P[N(t) = n]$$

As it is a decomposition, $P[N_1(t) = m/N(t) = n] = 0$ for $m > n$, thus:

$$P[N_1(t) = m] = \sum_{n=m}^{\infty} P[N_1(t) = m/N(t) = n]P[N(t) = n]$$

Since with each one of these n arrivals, N_1 is incremented by 1 with probability p_1, we have:

$$P[N_1(t) = m/N(t) = n] = \frac{n!}{m!(n-m)!} p_1^m \ (1-p_1)^{n-m}$$

thus:

$$\begin{aligned} P[N_1(t) = m] &= \sum_{n=m}^{\infty} \frac{n!}{m!(n-m)!} p_1^m \ (1-p_1)^{n-m} e^{-\lambda t} \frac{(\lambda t)^n}{n!} \\ &= (p_1 \lambda t)^m \frac{1}{m!} e^{-\lambda t} \sum_{n=m}^{\infty} \frac{1}{(n-m)!} (1-p_1)^{n-m} (\lambda t)^{(n-m)} \\ &= e^{-\lambda t} \frac{(p_1 \lambda t)^m}{m!} e^{(1-p_1)\lambda t} \\ &= e^{-p_1 \lambda t} \frac{(p_1 \lambda t)^m}{m!} \end{aligned}$$

Conclusion, N_1 is a Poisson process with rate $p_1\lambda$. N_1 being taken randomly, N_2 is also a Poisson process with rate p_2 by following the same reasoning.

Let us now check the independence of these two processes:

$$\begin{aligned} & P[N_1(t) = n_1, N_2(t) = n_2] \\ &= P[N_1(t) = n_1, N(t) = n_1 + n_2] \\ &= P[N_1(t) = n_1 / N(t) = n_1 + n_2] P[N(t) = n_1 + n_2] \\ &= \Big[\frac{(n_1+n_2)!}{n_1! n_2!} p_1^{n_1} (1-p_1)^{n_2} \Big] \ \Big[e^{-\lambda t} \frac{(\lambda t)^{(n_1+n_2)}}{(n_1+n_2)!} \Big] \\ &= e^{-p_1 \lambda t} \frac{(p_1 \lambda t)^{n_1}}{n_1!} \ \ e^{-(1-p_1)\lambda t} \frac{((1-p_1)\lambda t)^{n_2}}{n_2!} \\ &= P[N_1(t) = n_1] P[N_2(t) = n_2] \end{aligned}$$

These two processes are indeed independent.

It can be readily seen that these results can be extended to an independent dispatch of a Poisson process of rate λ over L destinations, with the

dispatching probability of p_i for the i-th destination, $i = 1, \ldots, L$, with $\sum_{i=1}^{L} p_i = 1$.

EXAMPLE 8.2.– A straightforward application is to model the routing process of a router. If we do not focus on the precise routing for particular addresses, but rather on macroscopic behavior, in terms of traffic dispatching, of the router, the whole routing process can be modeled by a process dispatching packet toward its L output interfaces according to some probability distribution $\{p_i\}_{i=1,\ldots,L}, \sum_{i=1}^{L} p_i = 1$. Each of the decomposed subtraffic will be Poisson with rate $p_i\lambda$.

8.6. Distribution of arrival instants over a given interval

Here, we are interested in the following problem. Given an interval of duration T, and knowing that it happens that there are N arrivals on this interval from a Poisson process. We are interested in the way the arrivals are positioned over the interval of duration T. The Poisson process being memoryless, we can take the interval $(0, T)$ without loss of generality.

Let us denote by $A(n, u, v)$ the following event: there are exactly n arrivals over the period (u, v). We will start with the case of only one arrival by showing that its arrival instant is uniformly distributed on $(0, T)$.

Let t be the arrival instant of the customer, take $a \in (0, T)$, the conditional distribution of t is written as:

$$\begin{aligned} P[t < a/A(1,0,T)] &= \frac{P[A(1,0,a) \text{ and } A(1,0,T)]}{P[A(1,0,T)]} \\ &= \frac{P[A(1,0,a) \text{ and } A(0,a,T)]}{P[A(1,0,T)]} \\ &= \frac{[e^{\lambda a}(\lambda a)][e^{\lambda(T-a)}]}{e^{\lambda T}(\lambda T)} = \frac{a}{T} \end{aligned}$$

We have $P[t < a/A(1,0,T)] = a/T$, we recognize an uniform distribution over $[0, T]$, we get thus the following result.

THEOREM 8.5.– Let N be a Poisson process. Knowing that over a period of duration T there is only one arrival over this period, then the arrival instant, denoted by t, of the customer is uniformly distributed over the period.

This result can be generalized to any given number of arrivals over a period of duration T, they will be uniformly distributed on $(0, T)$.

8.7. The PASTA property

One of the most interesting properties concerning the Poisson process arrivals is the one known as Poisson arrivals see time average (PASTA) which stipulates that, in steady state, the probability that an arrival from a Poisson process sees the system in a given state is the same as the probability that the system is in this same state at any given instant.

9

Markov Queueing Systems

This chapter is devoted to the queueing systems where all the events (departures and arrivals) take place in a Markov (exponential) way. We first present a generic framework concerning birth-and-death processes. This generic framework will then be used to study the queues of $M/M/.$ family, i.e. queues receiving Poisson arrival with a service distribution which is exponential.

9.1. Birth-and-death process

9.1.1. *Definition*

We refer to *state* of a system at a given instant t as being the set of parameters which characterize the system at that instant.

Generally, a stochastic system is characterized by its state which evolves in time. We consider here a very generic system (a black box) and we are interested solely by the evolution of the number of customers inside the box. Thus, the state of this system at time t is determined by the number of customers in the box at this instant, denoted by $S(t)$. We are interested here by the evolution in time of $S(t)$, knowing that the two sequences of events making $S(t)$ evolve are, respectively, arrivals (birth) and departures (death).

DEFINITION 9.1.– *A process* $\{S(t), t \geq 0\}$ *is a* birth-and-death[1] *process if it is in discrete states, i.e.* $\forall t \geq 0, S(t) \in \mathbb{N}$*, and verifies for any* $t > 0$ *and* $h > 0$:

1 It is also called a *birth-death* process.

Performance Evaluation by Simulation and Analysis with Applications to Computer Networks, First Edition. Ken Chen. © ISTE Ltd 2015. Published by ISTE Ltd and John Wiley & Sons, Inc.

1) *the processes of arrival and service are mutually independent, and their rates are independent of time*;

2) $P[\text{just 1 arrival within}]t, t+h]/S(t) = k] = \lambda_k h + o(h)$;

$P[\text{just 1 departure within}]t, t+h]/S(t) = k] = \mu_k h + o(h)$;

3) $P[0 \text{ arrival within}]t, t+h]/S(t) = k] = 1 - \lambda_k h + o(h)$;

$P[0 \text{ departure within}]t, t+h]/S(t) = k] = 1 - \mu_k h + o(h)$;

where λ_k (respectively, μ_k) is the arrival (respectively, departure) rate at the state k, i.e. when there is k customers in the system.

From this, it is readily deduced that:

– the probability of having more than one arrival during h is $o(h)$, idem for the number of departures;

– the probability of having simultaneously an arrival and a departure during h is also $o(h)$. Indeed, this probability is given by:

$$(\lambda_k h + o(h))(\mu_k h + o(h)) = \lambda_k \, \mu_k h^2 + o(h) = o(h)$$

– thus, any change of the system's state with a magnitude more than a unit is an event with probability of the order of $o(h)$.

We have thus (see section 14.3) a homogeneous, aperiodic and irreducible Markov process, which is such that from one state, the system can only evolve into its adjacent states.

REMARK 9.1.– The fact that an arrival rate can depend on the system's state is rather normal. At supermarket, one chooses to join the shorted cashier lane. In a telecommunication network, it is judicious, as far as possible, to carry out a load balancing by controlling traffic according to loads on respective routers.

9.1.2. *Differential equations*

Let us denote by $P_k(t)$ the probability $P[S(t) = k]$ which is that of having k customers in the system at time t. By enumerating all the possibilities, we obtain the following relations:

1) for $k > 0$:

$$\begin{aligned} P_k(t+h) = {} & P_k(t)[1 - \lambda_k h + o(h)][1 - \mu_k h + o(h)] \\ & + P_{k-1}(t)\lambda_{k-1} h + o(h) \\ & + P_{k+1}(t)\mu_{k+1} h + o(h) \end{aligned}$$

In this expression, the first term is related to the case where the system was in the state k and there was neither arrival nor departure during $(t, t+h]$, the second one the case where the system was in the state $k-1$ and there was exactly one arrival during $(t, t+h]$, and the third one the case where the system was in the state $k+1$ and there was exactly one departure during $(t, t+h]$;

2) for $k = 0$:

$$P_0(t+h) = P_0(t)[1 - \lambda_0 h + o(h)] + P_1(t)[\mu_1 h + o(h)] + o(h)$$

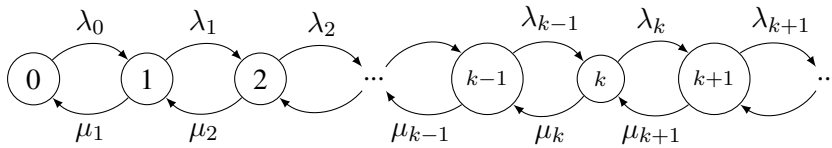

Figure 9.1. *Transition diagram of a birth-and-death process*

Figure 9.1, which is called a *transition diagram* (see section 14.3.6), gives these relations graphically. The latter can be simplified to:

– for $k > 0$:

$$\begin{aligned} P_k(t+h) - P_k(t) = {} & -(\lambda_k + \mu_k) P_k(t) h + \lambda_{k-1} P_{k-1}(t) h \\ & + \mu_{k+1} P_{k+1}(t) h + o(h) \end{aligned}$$

– for $k = 0$:

$$P_0(t+h) - P_0(t) = -\lambda_0 P_0(t) h + \mu_1 P_1(t) h + o(h)$$

By dividing the two sides of this system of equations by h then making $h \to 0$, we get:

$$\begin{cases} P_k^{'}(t) = -(\lambda_k + \mu_k)P_k(t) + \lambda_{k-1}P_{k-1}(t) + \mu_{k+1}P_{k+1}(t) & k \geq 1 \\ P_0^{'}(t) = -\lambda_0 P_0(t) + \mu_1 P_1(t) \end{cases}$$

We obtain a set of differential equations describing the dynamic behavior of the system.

EXAMPLE 9.1.– For certain cases, we can solve the preceding system of equations. Let us consider a pure birth process, defined by $\lambda_i = \lambda$ and $\mu_i = 0$. The preceding system of equations becomes:

$$\begin{cases} P_k^{'}(t) = -\lambda P_k(t) + \lambda P_{k-1}(t) & k \geq 1 \\ P_0^{'}(t) = -\lambda P_0(t) \end{cases}$$

With the initial condition $P_0(0) = 1$ (i.e. the system is empty at $t = 0$), one finds:

$$P_0(t) = e^{-\lambda t}$$

And by recurrence, we find:

$$P_k(t) = e^{-\lambda t}\frac{(\lambda t)^k}{k!}$$

We recognize a Poisson process with rate λ.

9.1.3. *Steady-state solution*

The previous example of a pure birth process is an exceptional case. In general cases, one cannot solve the system of differential equations. The main difficulty comes from the resolution of the differential equations, and thus of its dependence on time.

We thus seek a solution that is independent of time, which is known as *steady state* (or stationary) solution. When we have $\lim_{t\to\infty} P_k^{'}(t) = 0$, we are in steady state for large enough t. In what follows, we will suppose the existence of steady state. We will later seek conditions for the existence of steady state. These conditions are called *ergodicity* conditions.

In steady state, $\lim_{t\to\infty} P_k^{'}(t) = 0$, thus we have constant value $P_k = \lim_{t\to\infty} P_k(t)$, and we can write:

$$\begin{cases} (\lambda_k + \mu_k)P_k = \lambda_{k-1}P_{k-1} + \mu_{k+1}P_{k+1} & k \geq 1 \\ \lambda_0 P_0 = \mu_1 P_1 \end{cases}$$

These equations are called *balance equations* of the process. By observing that:

$$\forall k > 0, \quad \lambda_{k-1}P_{k-1} = \mu_k P_k,$$

and by proceeding in an iterative manner starting from P_1, we obtain expressions for $P_k, k > 0$, as a function of P_0:

$$P_k = \frac{\lambda_0\lambda_1 \ldots \lambda_{k-1}}{\mu_1\mu_2 \ldots \mu_k} P_0 = \left(\prod_{i=0}^{k-1} \frac{\lambda_i}{\mu_{i+1}} \right) P_0$$

The value of P_0 is got by $\sum_{k\in\mathbb{N}} P_k = 1$, i.e. the normalization equation:

$$P_0 = \frac{1}{1 + \sum_{k\geq 1} \prod_{i=0}^{k-1} \frac{\lambda_i}{\mu_{i+1}}} \qquad [9.1]$$

And thus we can deduce all the P_k from it.

However, it is not known *a priori* if the sum $\sum_{k\geq 1} \prod_{i=0}^{k-1} \frac{\lambda_i}{\mu_{i+1}}$ exists. Should this sum diverge, this would lead to $P_0 = 0$ and thus the nullity of all the P_k, we would thus have an *absurdity*.

In fact, it should be recalled that all these results are obtained under the assumption of the stationarity of the system. It is thus necessary to find the conditions, which are known as ergodicity conditions, which guarantee its existence. In practice, this means finding conditions ensuring $P_0 > 0$. Under the ergodicity conditions, one can calculate P_0, and consequently all the P_k. Physically speaking, $P_0 > 0$ means that the system becomes surely empty from time to time.

We can note that when the system is of finite size (say K), λ_i and μ_i for $i > K$ are null. We have thus a bounded sum in equation [9.1]; therefore,

P_0 exists *a fortiori*. This means that any system with a finite number of states reaches, in fine, the steady state.

This could lead to the conclusion that any existing physical system should have a stationary mode, since physical resources are intrinsically limited. Even if this reasoning is qualitatively (or, rather, philosophically) acceptable, it is necessary to see the speed of convergence and the calculability. Indeed, a system can be of finite but still large size, which leads to a very significant number of states. And the system can take a very long time before converging to its steady state, i.e. before one can draw some timely invariant estimation about the probability that the system should be at a given state. Thus, very often, for a system having a high amount of resources, it is more judicious to model it from a system having infinite states.

9.2. The $M/M/1$ queues

In this case, the arrival process is a Poisson process (M) and the service (departure) distribution is exponential (Markov, M). By taking the number of the customers in the queue (see Figure 9.2), denoted by X, as the state of the system, we can easily check that it is indeed a birth-and-death process. This observation is valid for all queues of the form $M/M/.$

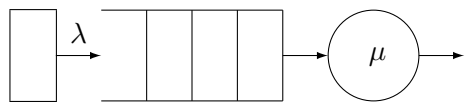

Figure 9.2. *An $M/M/1$ queue*

We can thus apply the general framework related to the birth-and-death process. The transition diagram of an $M/M/1$ queue is given by Figure 9.3. We have then:

$$\begin{cases} \lambda_k = \lambda, \, k \geq 0 \\ \mu_k = \mu, \, k \geq 1 \end{cases}$$

Let $\rho = \frac{\lambda}{\mu}$. ρ represents physically the load of the system, i.e. the ratio between submitted work and the processing capacity. Then, the P_k is given by:

$$P_k = P[X = k] = (\frac{\lambda}{\mu})^k P_0 = \rho^k P_0, \qquad k \geq 0$$

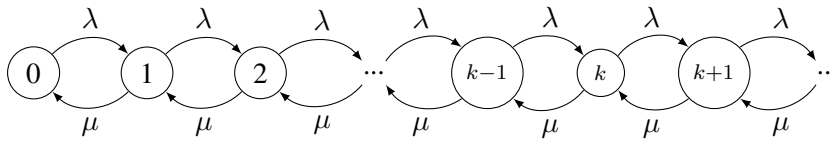

Figure 9.3. *Transition diagram of an $M/M/1$ queue*

We have $P_0 = (1 + \sum_{k=1}^{\infty} (\frac{\lambda}{\mu})^k)^{-1}$. After a simple calculation, we find:

$$P_0 = 1 - \rho$$

We deduce readily the stationarity condition: $\rho < 1$ so that $P_0 > 0$. Consequently, under this condition, we have, for any $k \geq 0$:

$$P_k = \rho^k (1 - \rho)$$

We can now calculate the mean number of customers in the system:

$$E[X] = \sum_{k=0}^{\infty} k P_k = (1 - \rho) \sum_{k=1}^{\infty} k \rho^k$$

We can note that $\sum_{k=1}^{\infty} k\rho^k$ is the value of the derivative of the function $f(x) = \sum_{k=0}^{\infty} x^k$, which is defined and differentiable for $|x| \leq 1, x \neq 1$, at the point $x = \rho$. Thus:

$$E[X] = \frac{\rho}{1 - \rho} \qquad [9.2]$$

Figure 9.4 shows a curve of $E[X]$ as a function of ρ.

The calculation of $E[X]$ can also be done through the probability generating function (PGF) of X (denoted by $g_X(z)$):

$$g_X(z) = \sum_{k=0}^{\infty} P_k z^k = \sum_{k=0}^{\infty} (\rho z)^k (1 - \rho) = \frac{1 - \rho}{1 - \rho z}$$

As $|\, z \,| \leq 1$ and $\rho < 1$, we thus find without difficulty:

$$E[X] = g_X(z)' \Big|_{z=1} = \frac{\rho}{1 - \rho} \qquad [9.3]$$

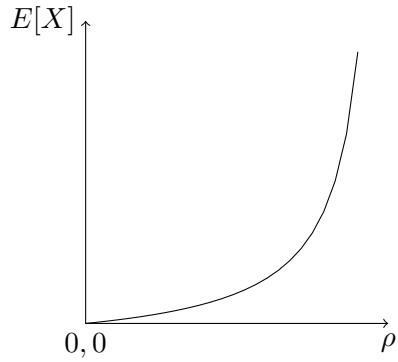

Figure 9.4. *Mean number of customers $E[X]$ versus load ρ in an $M/M/1$ queue*

Let us note T the sojourn time of a customer in the system (wait+service), by using Little's law ($E[X] = \lambda\, E[T]$), we have:

$$E[T] = \frac{\frac{1}{\mu}}{1-\rho} \qquad [9.4]$$

We present very hastily an important result called the *Theorem of Burke*. This theorem stipulates that the process at the output of an $M/M/1$ queue with input rate λ is a Poisson process with rate λ if the queue is in steady state (i.e. $\lambda < \mu$). In other words, we get the same process as the input.

9.3. The $M/M/\infty$ queues

In this case, there is a potentially infinite number of servers. It corresponds thus to a systems without waiting, i.e. there is always an available server for every new customer. This model is well suited for systems for which a resource oversizing is practiced. Here, we still have a birth-and-death process, with:

$$\begin{cases} \lambda_k = \lambda, & k \geq 0 \\ \mu_k = k\mu, & k \geq 1 \end{cases}$$

The transition diagram is given by Figure 9.5.

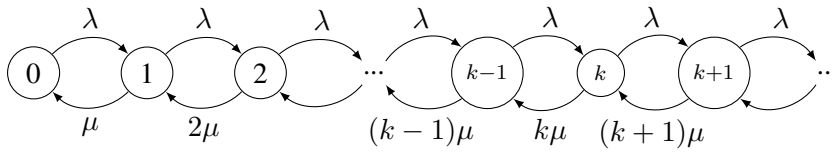

Figure 9.5. *Transition diagram of an $M/M/\infty$ queue*

Let $\rho = \frac{\lambda}{\mu}$. The stationary probabilities P_k are given by:

$$P_k = \frac{\rho^k}{k!} P_0, \qquad k \geq 1$$

Combining with $\sum_{k=0}^{\infty} P_k = 1$, we have:

$$P_0 = (\sum_{k=0}^{\infty} \frac{\rho^k}{k!})^{-1} = e^{-\rho}$$

$P_0 > 0$ for all ρ. The stationarity condition is always verified for any couple (λ, μ). This can be explained by the availability of an unlimited number of servers, ready to absorb any newly submitted work. We thus obtain:

$$P_k = \frac{\rho^k}{k!} e^{-\rho}, \qquad k \geq 1$$

The average number of customers in the systems ($E[X]$) and the average sojourn time in the system ($E[T]$) are calculated by using the same framework for birth-and-death systems. We have:

$$E[X] = \rho, \qquad E[T] = \frac{1}{\mu}$$

9.4. The $M/M/m$ queues

It is once more a birth-and-death process, with a unlimited buffer and up to m servers (see Figure 9.6):

$$\begin{cases} \lambda_k = \lambda, & k \geq 0 \\ \mu_k = \min\{k\mu, m\mu\}, & k \geq 1 \end{cases}$$

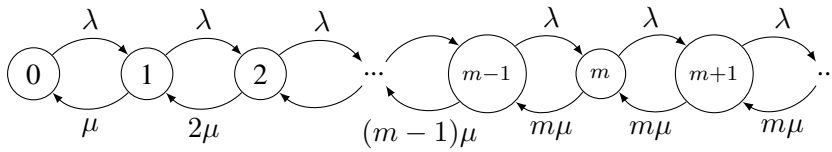

Figure 9.6. *Transition diagram of an $M/M/m$ queue*

The ergodicity condition is given by:

$$\rho = \frac{\lambda}{m\mu} < 1$$

The service rate having two different expressions depending on the value of k, the calculation of the P_k is thus divided into two parts:

– For $k \leq m$:

$$P_k = P_0 \prod_{i=0}^{k-1} \frac{\lambda}{(i+1)\mu} = P_0 (\frac{\lambda}{\mu})^k \frac{1}{k!}$$

– For $k > m$:

$$P_k = P_0 \prod_{i=0}^{m-1} \frac{\lambda}{(i+1)\mu} \prod_{i=m}^{k-1} \frac{\lambda}{m\mu} = P_0 (\frac{\lambda}{\mu})^k \frac{1}{m! m^{k-m}}$$

These expressions make it possible to calculate P_0 which is given by:

$$P_0 = \Big[1 + \sum_{k=1}^{m-1} (\frac{\lambda}{\mu})^k \frac{1}{k!} + \sum_{k=m}^{\infty} (\frac{\lambda}{\mu})^k \frac{1}{m! m^{k-m}}\Big]^{-1}$$

By using ρ ($\rho = \frac{\lambda}{m\mu}$), P_0 becomes:

$$P_0 = \Big[\sum_{k=0}^{m-1} (\frac{(m\rho)^k}{k!}) + (\frac{(m\rho)^m}{m!})(\frac{1}{1-\rho})\Big]^{-1}$$

From where we deduce the stationarity condition $\rho < 1$, i.e. $\lambda < m\mu$. Indeed, it is necessary that the load (λ) be supported by the maximal capacity of the system ($m\mu$).

We can thus calculate the waiting probability, i.e. the probability so that a new arrival discovers that all the m servers are occupied. This probability is:

$$P[\text{wait}] = P[X \geq m] = \sum_{k=m}^{\infty} P_k = \sum_{k=m}^{\infty} P_0 (\frac{(m\rho)^k}{m!}) \frac{1}{m^{k-m}}$$

This leads to:

$$P[\text{wait}] = \frac{(\frac{(m\rho)^m}{m!})(\frac{1}{1-\rho})}{\sum_{k=0}^{m-1}(\frac{(m\rho)^k}{k!}) + (\frac{(m\rho)^m}{m!})(\frac{1}{1-\rho})} \qquad [9.5]$$

In telephony, this formula is known as *Erlang's Formula C*, and is denoted by $C(m, \frac{\lambda}{\mu})$ or $E_{2,m}(\frac{\lambda}{\mu})$.

9.5. The $M/M/1/K$ queues

The reception capacity of this queue is limited to K customers. It is thus a system with loss. The transition diagram is given by Figure 9.7.

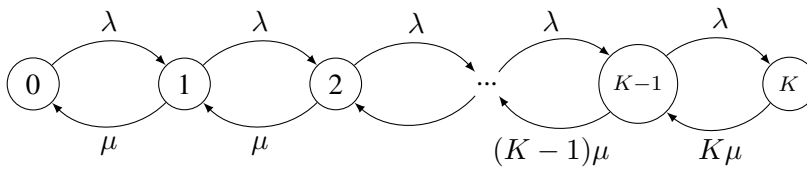

Figure 9.7. *Transition diagram of an $M/M/1/K$ queue*

We have:

$$\lambda_k = \begin{cases} \lambda, \, k < K \\ 0, \, k \geq K \end{cases}$$

$$\mu_k = \mu, \;\; k = 1, 2, \ldots, K$$

Like previously, we have:

$$P_k = \begin{cases} P_0 \prod_{i=0}^{k-1}(\frac{\lambda}{\mu}) = P_0(\frac{\lambda}{\mu})^k & k \leq K \\ 0 & k > K \end{cases}$$

and P_0 is given by:

$$P_0 = \frac{1 - \frac{\lambda}{\mu}}{1 - (\frac{\lambda}{\mu})^{K+1}}$$

It is easy to check that the ergodicity condition is always satisfied, which is foreseeable since we have a system with finite capacity.

9.6. The $M/M/m/m$ queues

This model corresponds to a system which guarantees an immediate service for each customer being admitted in the system; on the other hand, there is no waiting room: customers arriving when all the servers are occupied are simply discarded. Historically speaking, this model has been applied to switches in telephony. This model is also suitable for modeling a call center (hotline support) which does not keep the customers on standby if all the lines are occupied.

The transition diagram of this system is given by Figure 9.8, with:

$$\lambda_k = \begin{cases} \lambda, \, k < m \\ 0, \, k \geq m \end{cases}$$

$$\mu_k = k\mu, \;\; k = 1, 2, \ldots, m$$

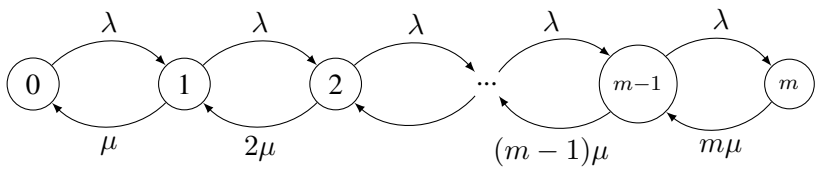

Figure 9.8. *Transition diagram of an $M/M/m/m$ queue*

As for $M/M/1/K$ (system with loss), the ergodicity condition is always satisfied. The P_k are given by:

$$P_k = \begin{cases} P_k = P_0 \prod_{i=0}^{k-1} \frac{\lambda}{(i+1)\mu} = P_0 (\frac{\lambda}{\mu})^k \frac{1}{k!} & k \leq m \\ P_k = 0 & k > m \end{cases}$$

and P_0 is given by:

$$P_0 = \left[\sum_{k=0}^{m} (\frac{\lambda}{\mu})^k \frac{1}{k!}\right]^{-1}$$

For this kind of system, it is important to know the loss ratio, i.e. the probability that a customer arrives at a moment when all the servers are occupied. This probability is simply P_m, which has as an expression:

$$P_m = \frac{(\frac{\lambda}{\mu})^m \frac{1}{m!}}{\sum_{k=0}^{m} (\frac{\lambda}{\mu})^k \frac{1}{k!}} \qquad [9.6]$$

P_m is known as *Erlang's Formula B*, and is denoted by $B(m, \frac{\lambda}{\mu})$ or $E_{1,m}(\frac{\lambda}{\mu})$. For information, this formula was found by Erlang in 1917.

9.7. Examples

9.7.1. *Two identical servers with different activation thresholds*

We consider a service system with two servers, S_1 and S_2, with identical processing capacity but different activation rules which are given hereafter:

– S_1 is always activated prior to S_2. In particular, it is activated as soon as there is at least one customer;

– S_2 becomes active if the number of customers is higher than a threshold value, denoted as l. As soon as the number becomes lower than l, S_2 ceases being active after the end of the current service.

This model is suitable for a class of systems with punctual heavy loads which one envisages to overcome by appointing a secondary server as a temporary reinforcement.

We suppose that the customers' arrival process is Poisson with rate λ and that the service time required by customers has an exponential distribution, with average duration of $1/\mu$ sec. We note that $\rho = \lambda/\mu$. The state of the system is the number of customers in the system, denoted as N and we note that $P_i = P(N = i)$:

1) Does $l = 1$ make sense? Why?

2) Can we compare this system to $M/M/2$? Why?

3) Under the stationarity assumption, give P_i in function of P_0 for $i \geq 1$.

4) Give the expression of the stationarity condition as a function of ρ and l.

5) Compare P_0 of this system with $l = 2$ against $P(N = 0)$ of an $M/M/2$.

6) Check if for $l = 30$, $\lambda = 1$ and $\mu = 5$ lead to a stable system.

Solution:

1) $l = 1$ does not make sense, because S_2 will not be activated if there is only one customer;

2) this system is reduced to $M/M/2$ if $l = 2$, in other cases, it is not;

3) we have a birth-and-death process with:

- $\lambda_k = \lambda$, $k \geq 0$
- $\mu_k = \mu$, $1 \leq k < l$ and $\mu_k = 2\mu$, $k \geq l$

From where, with $\rho = \lambda/\mu$:

- $P_i = \rho^i P_0$ for $0 \leq i < l$;
- $P_i = \frac{\rho^l}{2}(\frac{\rho}{2})^{i-l} P_0$ for $i \geq l$;

4) The stationarity condition is $0 < P_0 < 1$:

$$\sum_{i \in \mathbb{N}} P_i = P_0[\sum_{i=0}^{l-1} \rho^i + \frac{\rho^l}{2} \sum_{i \geq l} (\frac{\rho}{2})^{i-l}] = 1$$

thus:

$$P_0 = \frac{(1-\rho)(2-\rho)}{2-\rho-\rho^l}$$

The stationarity condition $0 < P_0$ implies $\rho < 1$;

5) For $l = 2$, we have:

$$P_0 = \frac{(1-\rho)(2-\rho)}{2-\rho-\rho^2}$$

It is the same as the one of $P(N = 0)$ for $M/M/2$ queue.

6) We have $\rho = 0.2 < 1$. The system is always stable.

9.7.2. *A cybercafe*

We consider a *cybercafe* (a café offering Internet access) offering n Web navigation terminals. Moreover, there is a waiting room of n places to accommodate customers waiting for a free terminal. It is noted that on average, the occupation time of a terminal is s time units, and that a customer leaves the cybercafe once he/she has finished using the terminal. Potential customers visit (arrive to) the cybercafe with a rate of λ customers per time unit. When all the terminals are taken but the waiting room is not full, only α percent of the potential customers still choose to enter to the cybercafe. If the waiting room is full, there are no new customers who will enter. It is supposed that the arrival process is Poisson and that the occupation time of terminal has an exponential distribution:

1) Calculate, as a function of the parameters (n, λ, α, s), the actual service rate μ_i and the arrival rate λ_i for the case where there are i customers in the cybercafe, $i = 0, \ldots, 2n$.

2) Calculate the average number of customers in the case of $n = 2$, $\lambda = 2$ per hour, $\alpha = 0.5$, $s = 0.5$ per hour; as well as the idle ratio (probability so that the cybercafe is empty).

Solution:

1) It is a birth-and-death system:

- $i = 0, \ldots, n-1, \lambda_i = \lambda$, $i = n, \ldots, 2n-1, \lambda_i = \alpha\lambda, \lambda_{2n} = 0$;
- $i = 1, \ldots, n-1, \mu_i = i/s$, $i = n, \ldots, 2n, \mu_i = n/s$.

2) Calculate at first the p_i, probability of having i customers, with $\lambda_i p_i = \mu_{i+1} p_{i+1}$ for $i > 0$:

- $\lambda_0 = \lambda_1 = \lambda = 2$, $\lambda_2 = \lambda_3 = \lambda/2 = 1$, $\lambda_4 = 0$;
- $\mu_1 = 1/s = 2$, $\mu_2 = \mu_3 = \mu_4 = 2/s = 4$;

3) We get from this:

- $p_1 = \frac{\lambda_0}{\mu_1} p_0 = p_0$;
- $p_2 = \frac{\lambda_1}{\mu_2} p_1 = p_0/2$;
- $p_3 = \frac{\lambda_2}{\mu_3} p_2 = p_0/8$;
- $p_4 = \frac{\lambda_3}{\mu_4} p_3 = \frac{1}{4} p_0/8 = p_0/32$;

4) With $\sum_{i=0}^{4} p_i = 1$, we have $p_0 = \frac{1}{1+1+1/2+1/8+1/32} = \frac{32}{85} = 0.3765$. The idle ratio of this cybercafe is thus 37.65%. Here is the average customer number $E[N] = \sum_{i=1}^{4} i p_i = 0.94$.

10

The $M/G/1$ Queues

This chapter is devoted to the $M/G/1$ queue and some of its variants (queues with server vacation, priority queueing). An important characteristic of the $M/G/1$ queues lies in the possibility of gaining some significant insights about the queue, such as the mean number of customers (and thus the mean sojourn time), with only the knowledge of the first and second moments of the service distribution, and without need to know its exact form. The submitted traffic must be a Poisson process. This kind of queue is thus suitable for modeling many systems of which one is unaware of the exact behavior, but only can estimate the *mean* and the *standard deviation* of the service time. This chapter also shows use of the transforms (Laplace, Z) which are commonly used in analytical modeling.

10.1. Introduction

In this chapter, we will study the $M/GI/1$ queue. It is fed by a Poisson process with rate λ. The customers are served by a single server, the service times are independent and identically distributed according to some distribution noted $S(x)$, its pdf will be noted by $s(x)$. In the what follows, we will note it $M/G/1$ by omitting I. Figure 10.1 illustrates such a queue.

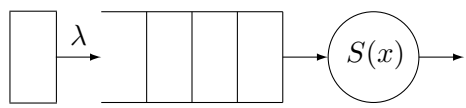

Figure 10.1. *An $M/G/1$ queue*

EXAMPLE 10.1.– Independent requests are submitted to a computation facility. If the submissions can be modeled roughly by a Poisson process, and if the required execution times have the same distribution, then the system can

Performance Evaluation by Simulation and Analysis with Applications to Computer Networks, First Edition. Ken Chen. © ISTE Ltd 2015. Published by ISTE Ltd and John Wiley & Sons, Inc.

be modeled by a $M/G/1$ queue where the service distribution is that of the execution time of the requests.

EXAMPLE 10.2.– IP packets are transmitted via an Ethernet interface at 100 Mbps. Packets have independent random sizes with the Pareto distribution. If the packet submission process can be modeled roughly by a Poisson process, then, the system can be modeled by a $M/G/1$ queue, where the service distribution is that of the duration of transmission which also has Pareto distribution.

10.2. Embedded Markov chain

We observe an $M/G/1$ queue at a given time t where we assume that a customer is being served. Generally, we are unable to predict the residual duration of the service (noted s_r). Indeed, even if we know the service distribution and consequently can draw a possible value for the *total* service duration (noted s_T), we do not have the knowledge of the starting instant of the service (noted t_b), i.e. the history of the service process, to be able to estimate $s_r = s_T - (t - t_b)$. Only in the case where $S(x)$ is Markov, which is memoryless, that the estimate of s_r becomes possible.

Now, let us observe the queue at particular instants where a customer leaves the system. We no longer need to worry about the history of the just finished service. As we can estimate the duration of the next service, as well as the number of customers arriving during the next service because the arrivals follow a Poisson process, we can estimate the number of customers remaining in the queue at the next departure, the very next occurrence of this particular sequence of instants. Let:

– Q_n = the number of customers remaining in the queue at the departure of the n-th customer (noted C_n);

– S_n = the service time of C_n;

– V_n = the number of customers arriving during the service of C_n.

We have:

$$Q_{n+1} = \begin{cases} Q_n - 1 + V_{n+1} & \text{for } Q_n > 0 \text{ (queue not empty)} \\ V_{n+1} & \text{for } Q_n = 0 \text{ (empty queue)} \end{cases}$$

Indeed, it is impossible to have $Q_{n+1} < Q_n - 1$, since there is only (and exactly) one less customer per departure. However, Q_{n+1} can take any value larger than $Q_n - 1$ due to the arrivals (V_{n+1}) during the service of the customer C_{n+1}.

As the service and arrival distributions are independent of the number of customers Q_n, V_n is also independent of the number of customers Q_n and depends solely on the service (of which the distribution is known) and the arrival process (which is Poisson). Consequently, if we know Q_n, we can estimate Q_{n+1}. The $\{Q_n\}$ thus form a Markov chain. It is qualified as *embedded Markov chain*, since it is defined only on a set of particular instants (departures). Let us consider the distribution of the duration between 2 successive transitions: There are two cases:

– if the queue is not empty ($Q_n > 0$), then the time until the next transition Q_{n+1} is simply the service time (S_{n+1}) of the customer (C_{n+1}) just being served, it has the service distribution $S(x)$;

– if the queue is empty ($Q_n = 0$), then the next transition Q_{n+1} occurs after arrival of a new customer and its service; the time until the next transition Q_{n+1} is thus the convolution of the distribution for Poisson process inter-arrival intervals ($1 - e^{-\lambda t}$) and the one of $S(x)$.

It has been shown that, since we have Poisson arrivals, the distribution obtained on these points is the same on any other instant (PASTA property). The chain $\{Q_n\}$ thus characterizes the $M/G/1$ queue. It is shown that the Markov chain $\{Q_n\}$ is aperiodic and irreducible.

10.3. Length of the queue

In this section, we study the $M/G/1$ in steady state. Let the r.v. X_Q denote the length of the queue, the r.v. X_V the number of customers arrived during a service, and the r.v. X_S a service duration, i.e.

$$\forall k \in \mathbb{N},\ \lim_{n\to\infty} E[Q_n^k] = E[X_Q^k],\ \lim_{n\to\infty} E[V_n^k] = E[X_V^k],$$

$$\lim_{n\to\infty} E[S_n^k] = E[X_S^k]$$

Let $Q(z)$ be the probability generating function of X_Q and $V(z)$ the one of X_V, we have: $Q(z) = \sum_{i=0}^{\infty} q_i z^i$ and $V(z) = \sum_{i=0}^{\infty} v_i z^i$. We will make use of transform techniques to deduce the *Pollaczek–Khinchin formula*. A practical interest of this formula consists of allowing the computation of mean queue length of (and so mean sojourn time in) $M/G/1$ queues, with the knowledge of:

– mean and variance of the service time;

– customers' arrival rate.

10.3.1. *Number of arrivals during a service period*

First, we will get the expression of $V(z)$ through the *Laplace transform* of the service distribution (see section 12.4.4.3).

Let us begin with v_k, which is the probability that there are exactly k new customers during a service,

$$\begin{aligned} v_k = P[X_V = k] &= \int_0^{\infty} P[v = k, x < s \leq x + dx]dx \\ &= \int_0^{\infty} P[v = k \,/\, s = x]s(x)dx \\ &= \int_0^{\infty} e^{-\lambda x} \frac{(\lambda x)^k}{k!} s(x)dx \end{aligned}$$

From where,

$$\begin{aligned} V(z) &= \sum_{k=0}^{\infty} v_k z^k = \sum_{k=0}^{\infty} \Big[\int_0^{\infty} e^{-\lambda x} \frac{(\lambda x)^k}{k!} s(x)dx \Big] z^k \\ &= \int_0^{\infty} e^{-\lambda x} \Big[\sum_{k=0}^{\infty} \frac{(\lambda z x)^k}{k!} \Big] s(x)dx = \int_0^{\infty} e^{-\lambda x} \Big[e^{\lambda z x} \Big] s(x)dx \\ &= \int_0^{\infty} e^{-\lambda(1-z)x} s(x)dx \end{aligned}$$

We recognizes the *Laplace transform* of $s(x)$, noted by $\tilde{S}(u)$, taken at $u = \lambda(1 - z)$. We thus get an important relation between the number of the arrivals during a service and the service time:

$$V(z) = \tilde{S}(\lambda(1 - z)) \qquad [10.1]$$

It is readily checked that we have:

$$V^{'}(z) = -\lambda \tilde{S}^{'}(\lambda(1-z))$$

Since $\tilde{S}^{'}(0) = -E[X_S] = -1/\mu$, thus:

$$V^{'}(1) = E[X_V] = \lambda E[X_S] = \frac{\lambda}{\mu} = \rho \qquad [10.2]$$

REMARK 10.1.– In fact, the relation [10.1] is valid for any duration of which we know the distribution through its *Laplace transform*.

10.3.2. *Pollaczek–Khinchin formula*

Let us note $q_i = P[X_Q = i]$, and write the Chapman–Kolmogorov equations:

$$\begin{cases} q_0 = q_0 v_0 + q_1 v_0 \\ q_1 = q_0 v_1 + q_1 v_1 + q_2 v_0 \\ \quad \dots \\ q_n = q_0 v_n + \sum_{i=1}^{n+1} q_i v_{n+1-i} \\ \quad \dots \end{cases}$$

Multiply by z^n, both sides of the equation with index n of this system ($q_n = q_0 v_n + \sum_{i=1}^{n+1} q_i v_{n+1-i}$), then make sum of all the equations, we get:

$$\begin{aligned} Q(z) &= q_0 V(z) + q_1 V(z) + q_2\, V(z) z + \dots + q_i V(z) z^{i-1} + \dots \\ &= V(z)\Big[q_0 + \sum_{i=1}^{\infty} q_i z^{i-1}\Big] = \frac{V(z)}{z}\Big[Q(z) - (1-z)q_0\Big], \end{aligned}$$

this gives:

$$Q(z) = \frac{(1-z)q_0 V(z)}{V(z) - z} \qquad [10.3]$$

For the computation of q_0, we notice that when $z = 1$, $Q(z) = 1$, the numerator and the denominator of the expression of right-hand side of equation

[10.3] tending both toward 0. We apply the rule of *L'Hôpital* at $z = 1$, which leads to:

$$1 = \frac{q_0}{1 - V^{'}(1)}$$

In fact, $V^{'}(1) = E[X_V] = \rho$, thus:

$$q_0 = 1 - \rho$$

and we have finally:

$$Q(z) = \frac{(1-\rho)(1-z)V(z)}{V(z) - z}$$

Thus we get the distribution of X_Q in function of that of X_V. By replacing $V(z)$ by $\tilde{S}(\lambda - \lambda z)$, we have:

$$Q(z) = \tilde{S}(\lambda - \lambda z)\frac{(1-\rho)(1-z)}{\tilde{S}(\lambda - \lambda z) - z} \qquad [10.4]$$

This formula is called the *Pollaczek–Khinchin formula* (P–K).

Let us take the derivative of $Q(z)$ and the value $Q^{'}(1)$, we obtain:

$$Q^{'}(1) = E[X_Q] = \rho + \frac{\lambda^2 E[X_S^2]}{2(1-\rho)} \qquad [10.5]$$

which gives the mean length of an $M/G/1$ queue. This formula is sometimes called the *Pollaczek–Khinchin mean value formula*, equation [10.4] is the *Pollaczek–Khinchin transform formula*. Notice that the mean length is determined entirely by the arrival rate and the first two moments of the service distribution. Formula [10.5] can be rewritten as:

$$E[X_Q] = \rho + \frac{\rho^2 + \lambda^2 Var[X_S]}{2(1-\rho)}$$

which highlights the fact that the mean length of the queue increases with the variance of the service distribution $Var[X_S]$.

EXAMPLE 10.3.– Here are two straightforward cases:

– For an $M/M/1$ queue, as $Var[X_S] = (1/\mu)^2$, we have:

$$E[X_Q] = \rho + \frac{\rho^2}{(1-\rho)} = \frac{\rho}{1-\rho}$$

– For an $M/D/1$ queue, $Var[X_S] = 0$, we have:

$$E[X_Q] = \rho + \frac{\rho^2}{2(1-\rho)} = \frac{\rho}{1-\rho} - \frac{\rho^2}{2(1-\rho)}$$

We can see that $E[X_Q]$ is smaller for an $M/D/1$ queue than for an $M/M/1$ queue, because the variance of service time of $M/D/1$ is null.

10.3.3. *Examples*

Below we give two examples of application of the $M/G/1$ queueing model.

EXAMPLE 10.4.– Consider a company which is connected to the Internet via a single router R through a physical connection L. The company is divided into 3 departments, denoted D_i, $i = 1, 2, 3$. Each department has its own communication facility, denoted C_i, $i = 1, 2, 3$, which are all attached to R. An analysis of the total flow received by R shows that there are several types of subflows $f_{i,j}$, $f_{i,j}$ being the subflow generated by an application of j class from department i. The rate of subflow $f_{i,j}$ will be denoted as $\lambda_{i,j}$. We suppose that these subflows are mutually independent and are Poisson. For the numerical applications, we give the following values:

– $\lambda_{1,1} = 30$ packets /s, $\lambda_{1,2} = 20$ packets /s, $\lambda_{1,3} = 15$ packets/s ;

– $\lambda_{2,1} = 5$ packets/s, $\lambda_{2,3} = 10$ packets/s ;

– $\lambda_{3,2} = 20$ packets/s.

1) What is the nature of the flow received by R? What is its rate, denoted λ?

2) It is supposed that the length of the packets has an exponential distribution with an average of 400 bytes/packet. Calculate the minimal bandwidth of L so that the response time of L (including both wait and transmission) is lower than 50 ms.

3) The same question, but instead suppose that the packets have a constant length of 200 bytes.

Solution:

1) The flow received by R is the composition of the flows coming from the three departments, these flows themselves are consisted of subflows. As these subflows are independent, the composed flow at the level of each department is a Poisson process which is independent of the two others. Then, the superposition of these three flows at R is also a Poisson process with rate $\lambda = \sum_i \sum_j \lambda_{i,j} = 100$ packets/s;

2) We have here an $M/M/1$ queue, whose latency is $\frac{1/\mu}{1-\lambda/\mu}$, with $\mu = C/(400 * 8)$ where C is the bandwidth of L. The stationarity condition leads to find μ such that:

$$\frac{1/\mu}{1-\lambda/\mu} < 0.05$$

which is $6/\mu \leq 0.05$ thus $\mu > 120$. So, $C \geq 6 * 400 * 8 * 20 = 960$ kbps. Notice that we must also have $\lambda/\mu < 1$, i.e. $\mu > \lambda = 100$, which is automatically verified since $\mu \geq 120$;

3) In this case, we have an $M/D/1$ queue whose latency is:

$$E[D] = \frac{1/\mu}{1-\lambda/\mu} - \frac{\lambda/(\mu)^2}{2(1-\lambda/\mu)}$$

Let $x = 1/\mu$, for $E[D] \leq 0.05$, we must have:

$$50x^2 - 6x + 0.05 \geq 0$$

Mathematically, we have the following solution: $x < 0.009$ or $x > 0.111$, i.e. $\mu > 111,1$ or $\mu < 9$. The second solution is to be rejected, because the stationarity condition claims $\mu > \lambda = 100$. Thus, $C > 111.1 * 400 * 8 = 355.5$ kbps.

EXAMPLE 10.5.– We consider a government office with computer-aided application processing facility. Applications are submitted at a rate of λ. We assume that the applications are submitted according to a Poisson process.

The applications are examined sequentially. Each application needs a random number of controls (for the sake of simplicity, the maximal number is here limited to 4). Its distribution (pmf) is given by Table 10.1. It is supposed that the time spent for each control is constant (noted r). We also assume that all the appliers are patient, i.e. they keep waiting without withdrawing from any application.

Control number	1	2	3	4
Probability	0.25	0.50	0.10	0.15

Table 10.1. *Distribution of the number of required controls*

1) Give the mean waiting time (the interval between the submission of an application and the beginning of its processing) as well as the condition to insure that this one does not diverge;

2) Give their numerical values with $r = 2$ min, $\lambda = 0.1$ application per minute;

3) There are more and more applications, and eventually the rate is multiplied by 5. To keep a reasonable waiting time, two options are offered:

i) A: enhance the processing capacity such that the processing time r will be divided by 5;

ii) B: open 4 additional offices which are identical to the current one, and to dispatch at random $1/5$ of the applications to each of the 5 offices.

Which option gives a shorter waiting time? Give and justify your choice.

Solution:

1) This is an $M/G/1$ queue. Denote the service time S. We have $E[S] = 2.15\,r$ and $E[S^2] = 5.55\,r^2$. Thus, $\rho = \lambda E[S]$. The stationarity condition is $\rho < 1$ and the mean waiting time W is given by:

$$W = \frac{\lambda E[S^2]}{2(1-\rho)}$$

2) Numerical results: $\rho = 0.43$, $W = 1.95$ minutes;

3) The choice is option A. We have to compute the two waiting times then make comparison.

i) For option B, at each office, $W_B = \frac{\lambda E[S^2]}{2(1-\rho)}$;

ii) For option A, $\lambda_A = 5\lambda$, $S_A = S/5$, thus:

$$\rho_A = \lambda_A E[S_A] = (5\lambda)(S/5) = \rho, \quad E[S_A^2] = E[(S/5)^2] = E[S^2]/25$$

So, $W_A = \frac{\lambda_A E[S_A^2]}{2(1-\rho_A)} = \frac{5\lambda E[S^2]/25}{2(1-\rho)} = W_B/5$. We can notice that by replacing 5 with an arbitrary number m, we still obtain the same type of conclusion.

10.4. Sojourn time

Let D_n denote the sojourn time of n-th customer noted as C_n. By definition, D_n is the duration of the interval between the arrival of C_n and its departure. D_n is the sum of two quantities: $D_n = W_n + S_n$ where W_n is the waiting time of C_n and S_n its service time.

Let us note respectively by X_D the sojourn time and X_W the waiting time in steady state, defined respectively by:

$$\forall k \in \mathbb{N},\ \lim_{n\to\infty} E[D_n^k] = E[X_D^k], \quad \forall k \in \mathbb{N},\ \lim_{n\to\infty} E[W_n^k] = E[X_W^k]$$

The distribution of X_D (respectively X_W) will be noted by $D(t)$ (respectively $W(t)$) and its pdf by $d(t)$ (respectively $w(t)$).

To determine $D(t)$, we notice that the number of customers remaining in the queue, X_Q, at the departure of a customer is by definition equal to the number of customers arriving during the sojourn of this very customer:

$$q_k = P[X_Q = k] = \int_0^\infty e^{-\lambda t} \frac{(\lambda t)^k}{k!} d(t)dt$$

with $Q(z) = \sum_{k=0}^{\infty} q^k z^k$, we get:

$$Q(z) = \sum_{k=0}^{\infty} \Big[\int_0^{\infty} e^{-\lambda t} \frac{(\lambda t)^k}{k!} d(t) dt \Big] z^k = \int_0^{\infty} e^{-\lambda t} \Big[\sum_{k=0}^{\infty} \frac{(\lambda z t)^k}{k!} \Big] d(t) dt$$

$$= \int_0^{\infty} e^{\lambda t z - \lambda t} d(t) dt = \int_0^{\infty} e^{-\lambda(1-z)t} d(t) dt = \tilde{D}(\lambda(1-z))$$

where $\tilde{D}(u)$ is the Laplace transform of $d(t)$. By using equation [10.1] between $V(z)$ and $\tilde{S}(u)$ with $u = \lambda(1-z)$, we have:

$$\tilde{D}(u) = \frac{\tilde{S}(u)(1-\rho)u}{\lambda \tilde{S}(u) + u - \lambda}$$

As we have $X_D = X_W + X_S$, we get:

$$\tilde{D}(u) = \tilde{W}(u)\tilde{S}(u)$$

Thus:

$$\tilde{W}(u) = \frac{(1-\rho)u}{\lambda \tilde{S}(u) + u - \lambda} \qquad [10.6]$$

In this way, we obtain, rather easily, the distributions of the sojourn time and the waiting time under the form of their Laplace transforms.

10.5. Busy period

The *busy period* is by definition the period during which the queue is not empty, i.e. duration during which the server is active. In a complementary way, the *idle period* is the period during which the system is empty. We focus here on the distribution $B(t)$ (with pdf $b(t)$) of the busy period X_B, as well as the distribution $R(t)$ of the idle period.

The calculus of $R(t)$ is quite simple: the idle period is equivalent to an interval during which there is no arrival. Thus, its distribution is identical to that of the inter-arrival intervals of a Poisson process:

$$R(t) = 1 - e^{-\lambda t} \qquad [10.7]$$

However, the calculus of $B(t)$ is much more difficult. Let us examine the following scenario. Let B_1 be the busy period starting with the arrival of the first customer, noted by C_1. We suppose that during the service of C_1, there is m newly arrived customers $C_1^1, \ldots, C_m^1$. The server has to serve all of these m new customers. However, there will be, in a similar manner, newly arrived customers during the service of each of them. B_1 will end only when all of these customers are served. As the number of customers induced by the service of a customer (m) is a random number, it is not obvious to find the duration of B_1.

This problem is solved by Takács. There is a detailed presentation in [KLE 75]. We report below only some of the main results.

Denoting by $\tilde{B}(u)$ the Laplace transform of the busy period B, we have:

$$\tilde{B}(u) = \tilde{S}[u + \lambda - \lambda\tilde{B}(u)]$$

This is a *functional* equation which is often difficult to solve. It was shown that its numerical resolution is possible by making the following iterative calculation:

$$\tilde{B}_{n+1}(u) = \tilde{S}[u + \lambda(1 - \tilde{B}_n(u))]$$

Indeed, with $0 \leq \tilde{B}_0(u) \leq 1$ and $\rho < 1$, we have $\lim_{n\to\infty} \tilde{B}_n(u) = \tilde{B}(u)$.

However, we can calculate the k-th moments (b_k) of X_B with $b_k = (-1)^k \tilde{B}^{(k)}(0)$. Let us calculate b_1 which gives the mean duration of a busy period:

$$b_1 = -\tilde{B}^{'}(0) = -[1 - \lambda\tilde{B}^{'}(0)]\,\tilde{S}^{'}[\lambda(1 - \tilde{B}(0))]$$

As $1 - \tilde{B}(0) = 0$ and $\tilde{S}^{'}(0) = -E[X_S]$, thus $b_1 = E[X_S](1 + \lambda b_1)$, which gives:

$$b_1 = \frac{E[X_S]}{(1 - \rho)}$$

Compared to an $M/M/1$ queue, we see that the mean value of the busy period of an $M/G/1$ queue is equal to the mean sojourn time in an $M/M/1$ queue having the same arrival and service rates.

10.6. Pollaczek–Khinchin mean value formula

We present here another approach to obtain the Pollaczek–Khinchin mean value formula. This approach uses an analysis based on mean values, as well as the concept of *residual service time*.

Consider the i-th customer (noted C_i) arriving on an $M/G/1$ queues. Let us introduce the following quantities related to C_i, by assuming the existence of their mean values:

– W_i its waiting time in the queue, with $W = E[W_i]$;

– X_i its service time, with $\overline{X} = E[X_i]$ and $\overline{X^2} = E[X_i^2]$;

– N_i the number of customers ($\{C_{i-1}, C_{i-2}, \ldots, C_{i-N_i}\}$) waiting for service in the queue when C_i arrives, with $N = E[N_i]$;

– R_i residual service time of the customer currently being served upon arrival of C_i, with $R = E[R_i]$.

The discipline of service being FIFO, it is clear that W_i is written as:

$$W_i = R_i + \sum_{j=i-N_i}^{i-1} X_j.$$

With the assumption of the existence of the mean values, and by using independence between the number of customers (N_i) and the service time (X_j), we have:

$$W = R + \overline{X}N.$$

By applying Little's law, we have $N = \lambda W$, and consequently:

$$W = \frac{R}{1-\rho}\,.$$

To evaluate R, notice (see Figure 10.2) that the residual service time, $R(t)$, jumps to the value of the service duration at the beginning of each service, then decreases to zero with a constant slope (-1).

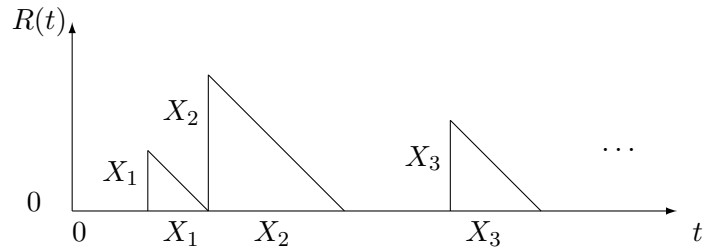

Figure 10.2. *Evolution of residual service time*

It is thus rather easy to evaluate $E[R]$. Consider an instant t such that the queue is empty at this moment, so, $R(t) = 0$. Let us denote by $N(t)$, the number of customers being served until t, the mean value of the residual service time at t, is given by:

$$\frac{1}{t}\int_0^t R(u)du = \frac{1}{t}\sum_{i=1}^{N(t)}\frac{1}{2}X_i^2$$

The right-hand term gives the surface under $R(t)$ and can be reformulated as follows:

$$\frac{1}{t}\sum_{i=1}^{N(t)}\frac{1}{2}X_i^2 = \frac{1}{2}\frac{N(t)}{t}\frac{\sum_{i=1}^{N(t)}X_i^2}{N(t)}$$

By supposing that the limits exist when t tends toward infinity, we have:

$$R = \frac{\lambda\overline{X^2}}{2}$$

From where the Pollaczek–Khinchin mean value formula for the waiting time:

$$W = \frac{\lambda\overline{X^2}}{2(1-\rho)} \qquad [10.8]$$

This approach by the analysis of mean values, in particular the mean residual service time, will be used for the study of queues with priorities and for the queues with vacation of the server.

10.7. $M/G/1$ queue with server vacation

Consider a token ring (see a short description in section 6.10), more particularly let us focus on the average access time of a station. We know that a station has the right to emit only if it has the token. By assuming that the arrivals constitute a Poisson process and the duration of each access is independent of the rest, we have an $M/G/1$ queue. However, here, the server is not always present: customers have to wait until the arrival of the server, i.e. the return of the token. This kind of queue is called a queue with *server vacation*.

Generally, a queue with server vacation is one whose server takes, at the end of a service (or a period of consecutive services), a period of vacation with a random duration, before coming back again.

Consider an $M/G/1$ queue where the server takes a period of vacation each time the queue becomes empty, i.e. vacation periods start at the end of busy periods. We assume that the vacation durations are i.i.d. r.v. Moreover, when the server returns from vacation and finds the system empty, it leaves again for another vacation period, and when the queue is not empty, it serves all the customers until the queue is empty.

To obtain the mean waiting time, the analysis follows the same framework that we used for an $M/G/1$ queue without server vacation. Indeed, for any particular customer, the vacation of the server only has the effect of delaying the availability of the server, we can thus consider the residual time of the vacation period in the same way as the residual service time (see Figure10.3). Thus, we always have $W = \frac{R}{1-\rho}$, but R must be re-evaluated accordingly, by including both the residual service time and the residual vacation time.

By definition, the vacation periods are inserted between the busy periods. As illustrated by Figure 10.3 where the arrow indicates the arrival of the third customer whose service has been delayed by the vacation (V_1) and will be started at the end of this vacation.

Let us denote by V_j the j-th vacation of the server, by supposing the existence of $\overline{V} = E[V_j]$ and $\overline{V^2} = E[V_j^2]$. Let us consider an instant t corresponding to the end of a vacation. Following the framework of the preceding section, and by letting $N(t)$ ($M(t)$) be the number of services respectively (respectively vacation) accomplished until the time t, we have:

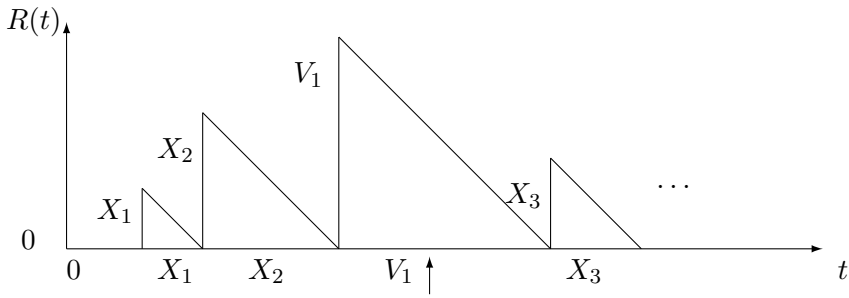

Figure 10.3. *Evolution of R of a queue with server vacation*

$$\frac{1}{t}\int_0^t R(u)du = \frac{1}{t}\sum_{i=1}^{N(t)}\frac{1}{2}X_i^2 + \frac{1}{t}\sum_{i=1}^{M(t)}\frac{1}{2}V_i^2$$
$$= \frac{1}{2}\frac{N(t)}{t}\frac{\sum_{i=1}^{N(t)}X_i^2}{N(t)} + \frac{1}{2}\frac{M(t)}{t}\frac{\sum_{i=1}^{M(t)}V_i^2}{M(t)}$$

By supposing the existence of the limits when $t \to \infty$, we get:

$$R = \frac{\lambda\overline{X^2}}{2} + \frac{1-\rho}{\overline{V}}\frac{\overline{V^2}}{2}$$

Recall that one vacation starts each time the queue is empty (with probability $1-\rho$) and that the mean duration of one vacation is $\overline{V}$, thus, the mean number of vacations is $\frac{1-\rho}{\overline{V}}$. The mean waiting time is given by:

$$W = \frac{\lambda\overline{X^2}}{2(1-\rho)} + \frac{\overline{V^2}}{2\overline{V}} \qquad [10.9]$$

We can see that the vacations increase the average waiting time by $\frac{\overline{V^2}}{2\overline{V}}$.

For information purposes, the service discipline considered here is known as *exhaustive* in the sense that the server serves *all* the customers before leaving

for a vacation, including those arriving *after* its return but *during* the current busy period.

An alternative consists of serving only the customers already present in the queue upon return of the server. Thus, when the server finishes serving the selected customer and leaves again for vacation, there could be customers left in the queue which have arrived *during* the current busy period. This discipline of service is known as *gated*, because everything occurs as if the server opened a door on its return to let the customers on standby enter, before closing it again just after.

Another alternative consists of serving a limited number of customers for each return. This discipline is known as *limited*. By assuming that each service time is itself bounded, this disciple has the advantage of offering an upper bound on the duration of a vacation, and, consequently the waiting delay. Consider the situation of a bandwidth sharing facility: the queues are networking interfaces and the server (resource) is the brute bandwidth. These interfaces are invited to transmitted in a cyclic manner. With the limited discipline, the access waiting time of each interface can be bounded. The analysis of this kind of systems is more complex [TAK 91].

10.8. Priority queueing systems

We consider a system admitting various types of customers which are characterized by their respective priorities. Naturally, the server serves the customers in the order of their priority. This mechanism of service with static priority is referred to as *Head Of the Line* (HOL) because the customers having the highest priority join the head of the queue. When a customer arrives while the server is serving a customer having lower priority, there are two options: either the server stops serving the current one by taking the newly arrived customer with higher priory, or the server continues to serve the current one then pick up the newly arrived one if it is still the most prior one. The former is known as *pre-emptive* and the latter *non-pre-emptive*.

An example of non-pre-emptive system is the transmission of an IP packet. Packets must be transmitted as a whole. An example of pre-emptive system is the interruption mechanisms in operating systems: an interruption instruction has to be handled prior to the currently running normal program.

Here, we consider only non-pre-emptive queues. In what remains, we will consider n classes of priority. For each class i ($1 \leq i \leq n$), we will denote by:

– λ_i the arrival rate;

– $\overline{S_i}$ the mean service time;

– $\overline{S_i^2}$ the second moment of the service time;

– $\rho_i = \overline{S_i}\lambda_i$ the load due to this class i;

– W_i the mean waiting time;

– T_i the mean sojourn time;

– N_i the mean number of customers from the class i in the queue.

The overall load of the queue will be denoted by ρ with $\rho = \sum_{i=1}^{n} \rho_i$. According to the general results about the $M/G/1$ queues, this overall load must verifies:

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

in order to insure that all the customers can be served. If not, only the customers of first k classes, which verify $\sum_{i=1}^{k} \rho_i < 1$, will be able to have an finite mean sojourn time. The customers of lower priorities will see their mean sojourn time tending to infinity, i.e. they will be blocked.

As we consider non-pre-emptive queues. Any customer must wait until the end of the current service, regardless of its priority. The residual service time, denoted by R, is given by:

$$R = \frac{1}{2} \sum_{i=1}^{n} \lambda_i \overline{S_i^2}$$

Now let us analyze the average waiting time of the class 1, the highest priority. A customer of this class has only to wait until all of the customers of the same class who arrived before him are being served, plus R, the residual service time it observes upon its arrival. From where:

$$W_1 = R + \overline{S_1} N_1$$

By applying Little's law ($N_1 = \lambda_1 W_1$), we have:

$$W_1 = R + \overline{S_1}\lambda_1 W_1$$

Written in another form, for class 1 we have:

$$W_1 = \frac{R}{1-\rho_1} \qquad [10.10]$$

For a customer of class 2, in addition to the waiting time caused by R and the service of the N_1 (respectively N_2) customers of class 1 (respectively class 2) already present in the queue upon its arrival, it also has to give way to all of the customers of class 1 who arrive after him but before the starting of its own service, i.e. all of the class 1 customers arrived during the *waiting time* of our target class 2 customer. On average, these customers are $\lambda_1 W_2$, W_2 being the duration of waiting of our customer. From where:

$$W_2 = R + \overline{S_1}(N_1 + \lambda_1 W_2) + \overline{S_2}N_2$$

Always by applying Little' law, we have:

$$W_2 = R + \rho_1 W_1 + \rho_1 W_2 + \rho_2 W_2.$$

By taking the formula of W_1 obtained previously, we have:

$$W_2 = \frac{R}{(1-\rho_1)(1-\rho_1-\rho_2)} = \frac{W_1}{1-\rho_1-\rho_2} \qquad [10.11]$$

By continuing the same reasoning, we obtain:

$$W_i = \frac{R}{(1-\sum_{k=1}^{i-1}\rho_k)(1-\sum_{k=1}^{i}\rho_k)} = \frac{W_{i-1}}{1-\sum_{k=1}^{i}\rho_k} \qquad [10.12]$$

This formula is known as the *formula of Cobram.*

11

Queueing Networks

This chapter is devoted to the queueing networks. We will see in particular that the arrival rate to each queue in such a system is obtained by solving a system of linear equations. We will also see that the model of the Jackson network offers a relatively simple way of dealing with queueing networks.

11.1. Generality

Consider a network of queues composed of M queues indexed from 1 to M. Figure 11.1 gives such a network with $M = 5$. For the sake of simplicity, the queues are not presented in their entirety but only represented by circles, the notations will be clarified below. There are two types of networks, open versus closed:

– An *open network* communicates with *outside*[1] (indexed by 0 for notation convenience). At the output of the queue i, a customer can move toward all the queues (including the queue i itself). If it moves toward outside (0), then it leaves the network definitively. If a customer comes from outside, then the network has a new arrival.

– A *closed network* does not communicate with outside. Consequently, there is always a constant number of customers circulating in the network.

EXAMPLE 11.1.– A telecommunications network can be modeled by an open network of queues where each node constitutes a queue while the users constitute the outside with submission and reception of packets.

1 Certain authors separate outside into two distinct parts, *source* (indexd by 0) and *sink* (indexed by ∞).

Performance Evaluation by Simulation and Analysis with Applications to Computer Networks, First Edition. Ken Chen. © ISTE Ltd 2015. Published by ISTE Ltd and John Wiley & Sons, Inc.

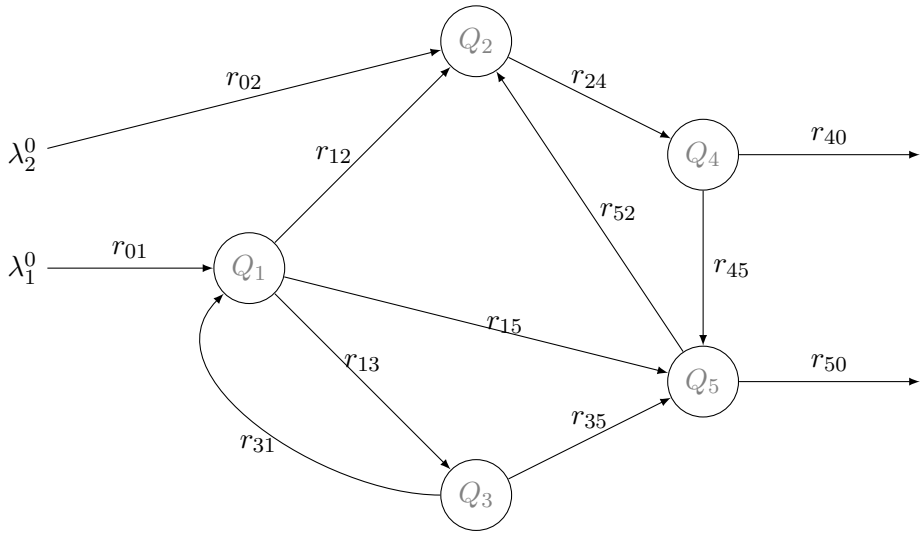

Figure 11.1. *A queueing network*

In the remainder, we denote the *transfer probabilities* when a customer leaves a queue as follows:

$$r_{ij} = P[\text{transfer from queue } i \text{ to queue } j]$$

We assume in the remaining part that each transfer is independent of any other and the transfer probabilities r_{ij} are constant in time. In addition, we assume that the transfer is carried out without delay.

The set of r_{ij} forms a matrix referred to as the *routing matrix* $\boldsymbol{R}$ which characterizes the transfers of customers inside the network

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1M} \\ r_{21} & r_{22} & \dots & r_{2M} \\ . & . & \dots & . \\ r_{M1} & r_{M2} & \dots & r_{MM} \end{bmatrix}$$

Similarly, we define:

$$r_{i0} = P[\text{leaves the network when leaving queue } i], \quad 1 \leq i \leq M$$

The following relation must be verified:

$$\forall i \in \{1, \ldots, M\},\ \sum_{j=1}^{M} r_{ij} + r_{i0} = 1 \qquad [11.1]$$

For a closed network, we have by definition $r_{i0} = 0$.

Let λ_i^0 be the rate of the arrivals coming from outside and arriving at queue i. Since there is no loss, flow must be preserved, and we then have the following relation:

$$\forall i \in \{1, \ldots, M\},\ \lambda_i = \lambda_i^0 + \sum_{j=1}^{m} r_{ji}\lambda_j \qquad [11.2]$$

where the left side of the equation represents the flow observed at the output of queue i and the right side represents the sum of the flows converging at queue i. Let us note:

$$\boldsymbol{\Lambda} = (\lambda_1, \ldots, \lambda_M) \quad \text{and} \quad \boldsymbol{\Lambda^0} = (\lambda_1^0, \ldots, \lambda_M^0)$$

We have:

$$\boldsymbol{\Lambda} = \boldsymbol{R\Lambda} + \boldsymbol{\Lambda^0}$$

which is, by denoting the identity matrix by $\boldsymbol{I}$:

$$(\boldsymbol{I} - \boldsymbol{R})\boldsymbol{\Lambda} = \boldsymbol{\Lambda^0} \qquad [11.3]$$

Thus, we have a system of linear equations. By solving it, we get the set of $\{\lambda_i\}_{i=1,\ldots,M}$.

Let n_i denote the number of customers in queue i, we take as the state of the network the vector:

$$\boldsymbol{n} = (n_1, n_2, \ldots, n_M)$$

The goal is to determine the joint probability, $p(\boldsymbol{n})$, that the system is in state $\boldsymbol{n}$.

11.2. Jackson network

We study here an important class of queueing network, the so-called *Jackson network*.

DEFINITION 11.1.– *A* Jackson network *is an open queueing network. For each queue i, its service distribution is Markov with rate μ_i and the first in first out (FIFO) service discipline. The customers arriving from outside follow a Poisson process with rate λ_i^0 and are independent of arrivals from outside to any other queue.*

As the processes of arrivals coming from outside are Poisson and are mutually independent, we can consider them as the decomposition of a single Poisson process with rate $\lambda = \sum_{i=1}^{M} \lambda_i^0$ and routing probability $r_{0i} = \frac{\lambda_i^0}{\lambda}$.

Despite the fact that the arrivals coming from outside are Poisson processes, we cannot consider the composed flow observed at the input of each queue as a Poisson process. Indeed, this flow is composed of subflows coming from other queues, and some customers may have already passed through the target queue. The Burke theorem (see section 9.2) cannot be applied. We do not have evidence that the composed flow is Poisson. Thus, we cannot consider individual queues of a Jackson network as $M/M/1$ queues, even if the service distribution is exponential.

We will see that, in the case of a *Jackson network*, it is possible to use, from the calculation point of view, the formulas of $M/M/1$. We will first establish the steady state equations. For that, we introduce the following notations:

– $\mathbf{1_i}$ is the vector which is null everywhere except on the i-th component which is worth 1, for example:

$$\boldsymbol{n} - \mathbf{1_i} = (n_1, n_2, \ldots, n_i - 1, \ldots, n_M);$$

– $\delta(n_i) = 1_{\{n_i > 0\}}$ indicates if the i-th component of the vector $\boldsymbol{n}$ is null or not.

Then, the balance equations in steady state are written as:

$$[\lambda + \sum_{i=1}^{M} \mu_i \delta(n_i)] p(\boldsymbol{n}) = \lambda \sum_{i=1}^{M} r_{0i} \delta(n_i) p(\boldsymbol{n} - \mathbf{1_i}) + \sum_{i=1}^{M} r_{i0} \mu_i p(\boldsymbol{n} + \mathbf{1_i}) + \sum_{i=1}^{M} \sum_{j=1}^{M} r_{ji} \mu_j \delta(n_i) p(\boldsymbol{n} + \mathbf{1_j} - \mathbf{1_i})$$

The term on the left side represents the departure rates from state $\boldsymbol{n}$, a departure from state $\boldsymbol{n}$ takes place either on arrival of a new customer to one of the queues (λ), or at the departure (toward outside or internal transfer) of a customer from a given queue ($\sum_{i=1}^{M} \mu_i$). The term on the right represents the arrival rate to state $\boldsymbol{n}$ which is composed of:

– arrivals coming from outside;

– departures toward outside;

– internal transfers.

The last term concerns a state change implying two queues; we can consider this case in steady state equations since the transfer is carried out without delay.

To find the probability $p(\boldsymbol{n})$, we notice that the preceding equation is verified by the following equality:

$$\text{for } n_i > 0 \qquad \lambda_i p(\boldsymbol{n} - \mathbf{1_i}) = \mu_i p(\boldsymbol{n})$$

Indeed, this equality implies in turn the following relation:

$$\text{for } n_i > 0 \qquad \lambda_j p(\boldsymbol{n} - \mathbf{1_i}) = \mu_j p(\boldsymbol{n} - \mathbf{1_i} + \mathbf{1_j})$$

by replacing $\boldsymbol{n}$ by $\boldsymbol{n}$-$\mathbf{1_i}$+$\mathbf{1_j}$.

Note $\rho_i = \frac{\lambda_i}{\mu_i}$ and developing the relation $\lambda_i p(\boldsymbol{n} - \mathbf{1_i}) = \mu_i p(\boldsymbol{n})$ then using it n_i times, we get:

$$\begin{aligned} p(\boldsymbol{n}) &= \rho_i p(n_1, n_2, \ldots, n_{i-1}, n_i - 1, n_{i+1}, \ldots, n_m) = \ \ldots \\ &= \rho_i^{n_i} p(n_1, n_2, \ldots, n_{i-1}, 0, n_{i+1}, \ldots, n_m) \end{aligned}$$

Repeating this operation for all of the queues, we get eventually:

$$p(\boldsymbol{n}) = (\prod_{i=1}^{M} \rho^{n_i}) p(\mathbf{0}) \qquad [11.4]$$

In conclusion, we only need to find $p(\mathbf{0})$, the probability that the system is empty, to be able to deduce from it all other $p(\boldsymbol{n})$. We use the normalization condition:

$$\sum_{\boldsymbol{n}} p(\boldsymbol{n}) = p(\mathbf{0})[\sum_{\boldsymbol{n}} (\prod_{i=1}^{M} \rho^{n_i})] = 1$$

At this stage, we introduce the stationarity condition:

$$\forall\ i = 1, \ldots, M,\ \ \rho_i < 1$$

This condition makes it possible to write:

$$\sum_{\boldsymbol{n}} (\prod_{i=1}^{M} \rho^{n_i}) = \prod_{i=1}^{M} \sum_{n_i=0}^{\infty} (\rho^{n_i}) = \prod_{i=1}^{M} (1 - \rho_i)^{(-1)}$$

Then, we get $p(\mathbf{0}) = \prod_{i=1}^{M} (1 - \rho_i)$ and consequently the following result.

THEOREM 11.1.– The probability that a Jackson network is in state $\boldsymbol{n}$ is given by:

$$p(\boldsymbol{n}) = \prod_{i=1}^{M} (1 - \rho_i) \rho_i^{n_i} \qquad [11.5]$$

We can note that this formula is in the so-called *product form*. Here, each term takes the same form as the expression for a $M/M/1$ queue which has, numerically, the same rates λ_i and μ_i. It should, however, be remembered that the term λ_i does not have the same signification as its counterpart in $M/M/1$, since generally the arrival process observed at the entry of a queue in the network is not Poisson. In addition, λ_i is not a parameter, it results from the resolution of a system of linear equations.

For each queue, the mean number of customers remaining in the queue is:

$$E[n_i] = \frac{\rho_i}{1 - \rho_i} \qquad [11.6]$$

The mean sojourn time, $E[T_i]$, is given by the Little law:

$$E[T_i] = \frac{1/\mu_i}{1 - \rho_i} \qquad [11.7]$$

It is thus possible to calculate the mean end-to-end delay for each possible path. In particular, if the "network" takes the form of a cascade of queues, the mean end-to-end delay is simply the sum of $E[T_i]$.

EXAMPLE 11.2.– We consider a processing center made up of five entities, respectively denoted as E_i, $i = 1, \ldots, 5$. We assume that the requests coming from outside can be submitted to each of these entities, e_i denoting the submission rate of the requests coming from outside to E_i ($i = 1, \ldots, 5$). The average processing time of a request on E_i is denoted by m_i. We denote by r_{ij} the probability that a request is transferred, after processing by E_i, toward E_j. By convention, E_0 indicates the outside of the system, i.e. r_{i0} gives the probability that a request is definitively processed and thus leaves the center. The following parameters are given:

– for $E_1 : e_1 = 2e, r_{12} = r_{15} = 1/3, r_{13} = r_{10} = 1/6$;

– for $E_2 : e_2 = 2e, r_{21} = r_{25} = 1/3, r_{24} = r_{20} = 1/6$;

– for $E_3 : e_3 = e, r_{34} = r_{35} = 1/3, r_{31} = r_{30} = 1/6$;

– for $E_4 : e_4 = e, r_{43} = r_{45} = 1/3, r_{42} = 1/6, r_{41} = 0$;

– for $E_5 : e_5 = 0, r_{51} = r_{52} = 1/6, r_{53} = r_{54} = 1/3$.

1) Propose a model of this system.

2) For $i = 1, \ldots, 5$, calculate the rate of the total arrivals, denoted by λ_i, that the entity E_i receives in steady state. For this calculation, $e = 1$ packet/s.

3) It is supposed from now on that the processing time of all the requests has an exponential distribution and that the traffic coming from outside are Poisson processes. In this case, what is the nature of this model?

4) Give the conditions to be respected by the m_i to be in steady state. For this calculation, $e = 1$ packet/s.

5) Determine the maximum value of m_5 so that the mean number of customers in E_5 does not exceed 20 s for $e = 1$ packet/s.

Solution:

1) We can model the system by a queueing network. A look on the parameters of the system shows that it is judicious to place E_5 in the center of a quadrilateral having for tops the four E_i, $i = 1, 2, 3, 4$. We can note, by examining the r_{ij}, that there is a symmetry between E_1 and E_2, as well as between E_3 and E_4. It would be useful to create a diagram (left as an exercise, see the example of Figure 11.1).

2) The equations are as follows

$$\begin{cases} \lambda_1 = & \lambda_2 r_{21} & +\lambda_3 r_{31} & & +\lambda_5 r_{51} & +e_1 \\ \lambda_2 = \lambda_1 r_{12} & & & +\lambda_4 r_{42} & +\lambda_5 r_{52} & +e_2 \\ \lambda_3 = \lambda_1 r_{13} & & & +\lambda_4 r_{43} & +\lambda_5 r_{53} & +e_3 \\ \lambda_4 = & \lambda_2 r_{42} & +\lambda_3 r_{34} & & +\lambda_5 r_{54} & +e_4 \\ \lambda_5 = \lambda_1 r_{15} & +\lambda_2 r_{25} & +\lambda_3 r_{35} & +\lambda_4 r_{45} & & \end{cases}$$

The symmetry between E_1 and E_2, along with that between E_3 and E_4, leads to $\lambda_1 = \lambda_2$ and $\lambda_3 = \lambda_4$, we have:

$$\begin{cases} \lambda_1 = & \lambda_1(1/3) & +\lambda_3(1/6) & & +\lambda_5(1/6) & +2e \\ \lambda_1 = \lambda_1(1/3) & & & +\lambda_3(1/6) & +\lambda_5(1/6) & +2e \\ \lambda_3 = \lambda_1(1/6) & & & +\lambda_3(1/3) & +\lambda_5(1/3) & +e \\ \lambda_3 = & \lambda_1(1/6) & +\lambda_3(1/3) & & +\lambda_5(1/3) & +e \\ \lambda_5 = (2/3)\lambda_1 & & +(2/3)\lambda_3 & & & \end{cases}$$

thus:

$$\begin{cases} \lambda_1 = \lambda_1(1/3) + \lambda_3(1/6) + \lambda_5(1/6) + 2e \\ \lambda_3 = \lambda_1(1/6) + \lambda_3(1/3) + \lambda_5(1/3) + e \\ \lambda_5 = (2/3)(\lambda_1 + \lambda_3) \end{cases}$$

This becomes:

$$\begin{cases} 4\lambda_1 - \lambda_3 - \lambda_5 = 12e \\ -\lambda_1 + 4\lambda_3 - 2\lambda_5 = 6e \\ -2\lambda_1 - 2\lambda_3 + 3\lambda_5 = 0 \end{cases}$$

After resolution, we have $\lambda_1 = \lambda_2 = \frac{42}{5}e$, $\lambda_3 = \lambda_4 = \frac{48}{5}e$, $\lambda_5 = 12e$.

3) It concerns a Jackson network.

4) m_i must respect $\rho_i < 1$. As $\lambda_i m_i = \rho_i$, so $m_i < \frac{1}{\lambda_i}$.

5) Jackson network: numerical value of $E[N_5]$ is given by, through applying the formula of corresponding $M/M/1$, $E[N_5] = \frac{\lambda_5 m_5}{1-\lambda_5 m_5}$. $E[N_5] < 20$ yields $12m_5 < 10 - 120m_5$, i.e. $m_5 < 13.2$. Thus, the maximum value of m_5 cannot exceed 13.2 s.

REMARK 11.1.– Here, the size of the problem is voluntarily limited to 5 in order to facilitate its manual resolution. In a professional context, the resolution of a system of linear equations can be done using adequate software. Thus, the approach presented through this example can be used to handle problems with much larger sizes.

11.3. Closed network

In this section, we will examine a closed network of queues. We always suppose that the service distribution is exponential with rate μ_i for queue i, $i = 1, \ldots, M$. There are primarily two major differences with the open network:

1) the number of customers in the network, denoted by N, is constant, i.e. $\sum_{i=1}^{M} n_i = N$. Consequently, the $\{n_i\}$ are dependent. The total number of possible states is equal to C_{N+M-1}^{M-1};

2) since there is no arrival coming from outside, it lacks a reference to normalize the rates λ_i.

The overall steady state equation is established in the same way:

$$\left[\sum_{i=1}^{M} \mu_i \delta(n_i)\right] p(\boldsymbol{n}) = \sum_{i=1}^{M} \sum_{j=1}^{m} r_{ji} \mu_j \delta(n_i) p(\boldsymbol{n} + \mathbf{1_j} - \mathbf{1_i})$$

Compared with that of the Jackson network, we can note the absence of the terms related to the outside. We always have:

$$p(\boldsymbol{n}) = C \left(\prod_{i=1}^{M} \rho^{n_i} \right)$$

where C is a constant (similar to the $p(\mathbf{0})$ for the open networks). However, it must be considered as a normalization constant, and not as the probability of being in the empty state (which does not exist). So it is not as easy, contrary to the case of Jackson network, to find C. Indeed, we cannot exploit the normalization condition:

$$\sum_{\boldsymbol{n}} p(\boldsymbol{n}) = C \left[\sum_{\boldsymbol{n}} \left(\prod_{i=1}^{M} \rho^{n_i} \right) \right] = 1$$

in the same manner as previously, because we cannot vary n_i separately.

It can be proved that the final result is put in the following form:

$$p(\boldsymbol{n}) = \frac{1}{G(N, M)} \prod_{i=1}^{M} (\rho_i)^{n_i} \qquad [11.8]$$

where $G(N, M)$ is the factor of normalization defined by:

$$G(N, M) = \sum_{\boldsymbol{n}} \prod_{i=1}^{M} (\rho_i)^{n_i}$$

$G(N, M)$ is obtained by the following iterative formula:

$$n \geq 1,\ m \geq 1,\ G(n,m) = G(n,m-1) + \rho_m G(n-1,m) \qquad [11.9]$$

with the following initial conditions:

$$\begin{cases} G(n,1) = \rho_1^n \ n = 0,1,\ldots,N \\ G(0,m) = 1 \ \ m = 1,\ldots,M \end{cases}$$

PART 3

Probability and Statistics

12

An Introduction to the Theory of Probability

The theory of probabilities aims to study random phenomena. We provide here a short introduction to some of its basic concepts. The latter is used in a recurring way in this book, implicitly or explicitly. For an in-depth study, we have to consult specialized books, for example the classical [FEL 68], or [DEG 81] which gives a combined presentation of probability and statistics. [ALL 78] is also a good introducory book.

12.1. Axiomatic base

12.1.1. *Introduction*

12.1.1.1. *Sample space*

A random phenomenon produces results which are by definition not predictable. The set of all possible outcomes is however quite identifiable in general. Taking the classical example of "Coin Tossing": each trial produces an outcome (*head* or *tail*) which cannot be known in advance; on the contrary, we are sure that the results (outcomes) are either *head* or *tail* and nothing else.

The result of each trial is called an *outcome*. A *sample space*, generally noted by Ω, is the set of all the possible outcomes. For example, if we are interested by the occupancy of a buffer with a capacity of 100 packets, $\Omega = \{0, 1, \ldots, 100\}$. Each observation on the buffer is an outcome $o \in \Omega$ giving the buffer's current occupancy.

12.1.1.2. *Events*

Let us consider again the example of buffer occupancy. Generally, we are not interested by each specific occupation, say, $o = 10$ packets, but by some

Performance Evaluation by Simulation and Analysis with Applications to Computer Networks, First Edition. Ken Chen. © ISTE Ltd 2015. Published by ISTE Ltd and John Wiley & Sons, Inc.

phenomena considered as relevant. For instance, $o < 25$ may be considered as a weak load situation, whereas $o > 75$ can be considered as a prelude to overflow.

An *event*, often noted by a capital letter, is a subset of Ω, gathering one or more outcomes. Let A be an event, $A \subset \Omega$. In other words, $A \in \mathcal{P}(\Omega)$ where $\mathcal{P}(\Omega)$ is the partition of Ω. Here are some examples:

– With "coin tossing" ($\Omega = \{\text{head, tail}\}$), we can focus only on the event $A = \{head\}$.

– A router has three networking interfaces $\Omega = \{\text{if0, if1, if2}\}$. One event could be $A = \{\text{if0}\}$, i.e. the fact that a packet is routed to "if0".

– Some special cases:

- $\emptyset$ symbolizes an "event" which never takes place, i.e. an impossibility;

- let ω be a particular outcome of Ω ($\omega \in \Omega$), its occurrence is an event: $E_\omega = \{\omega\}$;

- Ω is also an "event", and it always occurs, i.e. it symbolizes a certainty.

12.1.1.3. *Probability*

Here, we say simply that the probability is a *measure* which gives a value (a real ranging in $(0, 1)$) to each possible event (and *a fortiori* to each outcome which is a particular event). It can be viewed as a kind of quantification of the occurrence *frequency* of the event.

12.1.2. *Probability space*

We presented a summary and intuitive introduction which proposes the following three concepts:

– identification of all of the possible outcomes;

– identification of all of the *interesting* events;

– quantification of these events.

We present here briefly a formalization of these concepts, which constitute the axiomatic basis of the theory of probability.

12.1.2.1. *σ-Algebra*

When we are interested by a certain event (say E), we have to be concerned with the opposite event $\overline{E}$, i.e. the event that E does not occur. Moreover, when we are interested in, individually, a certain number of events, we are also concerned with the occurrence of one of them. These observations lead to the definition of the σ-Algebra.

DEFINITION 12.1.– A σ-Algebra, noted $\mathcal{F}$, defined on a sample space Ω is a family of subsets of Ω such that:

– $\emptyset \in \mathcal{F}$.

– If $A \in \mathcal{F}$, then $\overline{A} \in \mathcal{F}$ (in particular $\Omega \in \mathcal{F}$) .

– If a sequence of subsets $\{A_n\}_{n\geq 1}$ is such that $A_n \in \mathcal{F}$ for each $n \geq 1$, then $(\cup_{n=1}^{\infty} A_n) \in \mathcal{F}$.

EXAMPLE 12.1.– Taking $\Omega = \mathbb{R}$, the *Borel σ-Algebra*, noted as $\mathcal{B}(\mathbb{R})$, is the smallest σ-Algebra containing all of the open intervals.

12.1.2.2. *Probability measure*

DEFINITION 12.2.– Let Ω be a sample space and $\mathcal{F}$ a σ-Algebra defined on Ω, a probability measure defined on $(\Omega, \mathcal{F})$, noted P, is a real-value function, a mapping from $\mathcal{F}$ to $\mathbb{R}$, such that:

– $\forall A \in \mathcal{F}, P(A) \in [0, 1]$.

– $P(\Omega) = 1$.

– If $A_n \in \mathcal{F}$, $n \geq 1$, is a sequence of mutually exclusive events ($A_i \cap A_j = \emptyset$ if $i \neq j$), then:

$$P(\cup_{n=1}^{\infty} A_n) = \sum_{n=1}^{\infty} P(A_n).$$

The triple $(\Omega, \mathcal{F}, P)$ constitutes a *probability space*. If $P(A) = 0$, A is a *zero-probability* event. If $P(A) = 1$, A is an *almost sure* (a.s.) event, we also say that A occurs *almost surely*.

12.1.2.3. *Basic properties*

We give below some elementary properties of a probability measure which result (almost) directly from its definition:

– $P(\emptyset) = 0$.

– $P(A) = 1 - P(\overline{A})$.

– $P(A \cup B) = P(A) + P(B) - P(A \cap B)$.

– If $A \subset B$, then $P(A) \leq P(B)$.

12.1.3. *Set language versus probability language*

The events are subsets of Ω and receive a value as their probability measure. In Table 12.1, we provide a correspondence between the set language and the probability language.

Set	Probability
$A \cup B$	A **or** B occurs
$A \cap B$	A **and** B occur
$\overline{A}$	A **does not** occur
$A \subset B$	(the occurrence of) A **implies** (the one of) B
$A \cap B = \emptyset$	A and B are **mutually exclusive**

Table 12.1. *Correspondence between the set language and the probability language*

12.2. Conditional probability

12.2.1. *Definition*

DEFINITION 12.3.– The *conditional probability* of event A given B, noted $P(A/B)$, is defined by:

$$P(A/B) = \frac{P(A \cap B)}{P(B)} \qquad [12.1]$$

This formula is known as *Bayes's formula*. We deduce from it immediately that if A and B are mutually exclusive (disjointed), i.e. $A \cap B = \emptyset$, then $P(A/B) = 0$.

12.2.2. *Law of total probability*

Let $\{A_i\}_{i\in\mathbb{N}}$ be a finite or countably infinite partition of Ω, i.e. $(\cup_{i\in\mathbb{N}} A_i) = \Omega$ and $A_i \cap A_j = \emptyset$ for $i \neq j$. Let A be any event, then:

$$P(A) = \sum_{i=1}^{N} P(A/A_i)P(A_i) \qquad [12.2]$$

This formula, called the *formula of total probability* (and also the *law of total probability*), is very useful. Indeed, it facilitates to partition the computation of the probability of an event into several specific cases (A_i, with probability $P(A_i)$). If the condition A_i does help to calculate $P(A/A_i)$, it suffices to complete it by the calculation of $P(A_i)$, and this makes the calculation of $P(A)$ easier.

12.3. Independence

DEFINITION 12.4.– Two events A and B are *independent* if and only if:

$$P(A \cap B) = P(A)P(B). \qquad [12.3]$$

We can give the following remarks:

– If A and B are independent, the conditional probability:

$$P(A/B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A)P(B)}{P(B)} = P(A),$$

in other words, the event B has no impact on A. This corresponds well to the intuitive notion of independence in our everyday life language.

– We should not confuse mutual exclusion and independence. The former indicates in fact implicitly a strong "dependence". If A and B are mutually excluded, we have $P(A \cap B) = 0$. If, moreover, A and B are independent, we must at least have $P(A) = 0$ or $P(B) = 0$. i.e. at least one of them should never occur (zero-probability event).

12.4. Random variables

12.4.1. *Definition*

Very often, an event can be described by a certain quantity (integer or real). For example, the state of a switch, if we are interested in the phenomena of congestion, can be described by the number of packets, say N, in its buffer. With N, the event "congestion" is simply the event "N exceeds a certain threshold". As N does not have a fixed value, it is called a *random variable*, which is often indicated under the abbreviation r.v. Roughly speaking, a random variable (r.v.) is a mapping of a certain outcome of Ω in the space of $\mathbb{R}$.

DEFINITION 12.5.– Let $(\Omega, \mathcal{F}, P)$ be a probability space. A *random variable*, noted X, is a measurable function mapping Ω to $\mathbb{R}$, such that for all interval $I \subset \mathbb{R}$, $X^{-1}(I) \in \mathcal{F}$.

By means of the random variables, we can thus transpose any probability space $(\Omega, \mathcal{F}, P)$ into $\mathbb{R}$ along with its Borel σ-algebra ($\mathcal{B}(\mathbb{R})$) and with a certain probability measure which defines the *distribution* of the variable.

Rigorously speaking, an r.v. is a quantification of the trials and should be consequently expressed in the form $X(\omega)$, with $\omega \in \Omega$. By convention, we denote an r.v. with a capital letter and by omitting ω, $\{X = r\}$ corresponds to the event $\{\omega / X(\omega) = r\}$.

12.4.2. *Cumulative distribution function*

The *cumulative distribution function* (CDF) offers one of the means to characterize a random variable.

DEFINITION 12.6.– The cumulative distribution function of an r.v. X, noted $F_X(x)$, is given by:

$$F_X(x) = P(X \leq x) \qquad [12.4]$$

When there is no ambiguity, one can omit the index X. The cumulative distribution functions have the following properties in common:

– $F(x) \in [0,\ 1]$;

– $F(x)$ is an increasing function;

– $F(x)$ is right continuous.

Subsequently, due to the difference in mathematical formulation, we will handle separately continuous r.v. and discrete r.v.

12.4.3. *Discrete random variables*

12.4.3.1. *Definition*

DEFINITION 12.7.– An r.v. X is *discrete* if it takes values in a finite or countably infinite set E:

$$X \in E,\ E = \{x_i\}_{i \in \mathbb{N}}$$

In statistics, a discrete r.v. is also termed as *categorial*.

The distribution of a discrete r.v. X is determined by its *probability mass function* (pmf) $\{p_i\}_{i \geq 0}$ such as:

$$\forall i \geq 0,\ \ P(X = x_i) = p_i$$

We can imagine each p_i as being a weight (a mass) associated with the value x_i.

A particular and very frequent case is that of a discrete r.v. taking its values in $\mathbb{N}$, i.e. $E = \mathbb{N}$.

12.4.3.2. *Mathematical expectation*

Let X be a discrete r.v. taking values in $E = \{x_i\}_{i \geq 0}$ and with distribution $\{p_i\}_{i \geq 0}$. The *mathematical expectation* of X, noted $E[X]$, is given by (under existence condition):

$$E[X] = \sum_{i=0}^{\infty} x_i\ p_i \qquad [12.5]$$

The *variance* of X, noted $Var[X]$, is given by (under existence condition):

$$Var[X] = E[(X - E[X])^2], \qquad [12.6]$$

which can be readily rewritten as:

$$Var[X] = E[X^2] - (E[X])^2. \quad [12.7]$$

The mathematical expectation is also called *expectation* or *mean*, in the rest, we will use these shorter forms.

$E[X]$ is also called the *first moment* of X, $E[X^2]$ is the *2nd moment* of X. More generally, $E[X^k]$ is the k-th moment of X.

12.4.3.3. *Probability generating function*

Let X be a discrete r.v. taking values in $E = \{x_i\}_{i\geq 0}$ and with distribution $\{p_i\}_{i\geq 0}$. Its associated *probability generating function*, noted $g_X(z)$, is defined by:

$$g_X(z) = \sum_{i\in\mathbb{N}} p_i z^i, \quad |z| \leq 1$$

The probability generating functions (PGF) have the following properties:

– $g_X(1) = 1$;

– $g_X^{'}(1) = E[X]$;

– $g_X^{''}(1) + g_X^{'}(1) = E[X^2]$.

These relations allow the calculation of the first two moments through probability generating function.

It is fundamental that X be an *integer* r.v. Indeed, the aforementioned properties are no longer true otherwise.

12.4.4. *Continuous random variables*

12.4.4.1. *Definition*

DEFINITION 12.8.– An r.v. X is *continuous* if it takes values in $\mathbb{R}$ and such that $\forall a \in \mathbb{R}, P(X = a) = 0$.

This definition shows clearly the difference of a continuous r.v. with respect to (w.r.t.) a discrete r.v.in the sense that no mass can be cumulated on any single value for a continuous r.v.

The distribution of X is determined by its *probability density function* (pdf, often written in minuscule letters), denoted as $f_X(.)$, which is given by:

$$f_X(x) = F_X^{'}(x).$$

12.4.4.2. *Mathematical expectation*

Let X be a continuous r.v. with pdf $f(x)$. The *mathematical expectation* of X, noted $E[X]$, is given by (under existence condition):

$$E[X] = \int_{x=-\infty}^{\infty} xf(x)dx \qquad [12.8]$$

The *variance* of X ($Var[X]$), as well as any of its k-th moments ($E[X^k]$), are defined in the same manner as that of a discrete r.v. In particular,

$$Var[X] = E[(X - E[X])^2] = E[X^2] - (E[X])^2. \qquad [12.9]$$

12.4.4.3. *The Laplace transform*

For each positive continuous r.v., say X, we can define its *Laplace transform*, noted $\tilde{X}(u)$, which is given by:

$$\tilde{X}(u) = E[e^{-uX}] = \int_{x=0}^{\infty} e^{-ux} f(x)dx \qquad [12.10]$$

Under the existence condition of $E[X^k]$, we can show that:

$$\left.\frac{d^{(k)}\tilde{X}(u)}{du^k}\right|_{u=0} = (-1)^k E[X^k] \qquad [12.11]$$

In particular:

$$(\tilde{X})^{'}(0) = -E[X], \qquad \text{and} \qquad (\tilde{X})^{''}(0) = E[X^2] \qquad [12.12]$$

12.4.5. *Characteristic function*

The *characteristic function* of an r.v. X, noted $\phi_X(u)$, is given by:

$$\phi_X(u) = E[e^{iuX}] \qquad [12.13]$$

which is expressed as follows, depending on the nature of X:

– X is a discrete r.v., with pmf $\{p_n = P(X = x_n)\}_{n\in\mathbb{N}}$:

$$\phi_X(u) = E[e^{iuX}] = \sum_{n\in\mathbb{N}} e^{iux_n} p_n \qquad [12.14]$$

– X is a continuous r.v., with pdf $f_X(x)$:

$$\phi_X(u) = E[e^{iuX}] = \int_{x\in\mathbb{R}} e^{iux} f_X(x)dx \qquad [12.15]$$

In this latter case, $\phi_X(u)$ can be viewed as the *Fourier transform* of $f_X(x)$. The latter can be calculated from $\phi_X(x)$ by:

$$f_X(x) = \frac{1}{2\pi} \int_{u\in\mathbb{R}} e^{-iux} \phi_X(u)du$$

12.5. Some common distributions

We give below some common probability distributions. The notation $X \sim \mathcal{L}$ means X has the $\mathcal{L}$ distribution.

12.5.1. *Bernoulli distribution*

An r.v. X has Bernoulli distribution with parameter p, noted $B(p)$, if X takes values in $\{0, 1\}$ with:

$$\begin{cases} P(X = 1) = p \\ P(X = 0) = 1 - p = q \end{cases} \qquad [12.16]$$

We have:

$$E[X] = p, \qquad Var[X] = pq \qquad [12.17]$$

The Bernoulli distribution can be used to model many phenomena. *Coin tossing* is a classical example of Bernoulli distribution. In telecommunications, this distribution can be a model for the loss phenomena due to *punctual* transmission channel noise.

12.5.2. *Binomial distribution*

An r.v. X has a *binomial distribution* with parameters (n, p), noted $B(n, p)$, if $X \in \{0, \ldots, n\}$ with:

$$\forall k, 0 \leq k \leq n, \quad p_k = P(X = k) = C_n^k p^k q^{n-k} \qquad [12.18]$$

where $q = 1 - p$. We have:

$$E[X] = np, \qquad Var[X] = npq. \qquad [12.19]$$

Consider a sequence of n i.i.d. r.v., $\{X_i\}_{i=1,\ldots n}$ with $P(X_i = 1) = p$, i.e. X_i has Bernoulli distribution $B(p)$. Consider the r.v. $X = \sum_{i=1}^{n} X_i$. It can be shown that $X \sim B(n, p)$. A binomial r.v. can thus be considered as the result (sum) of n i.i.d. Bernoulli r.v.

The binomial distribution has various applications. Consider a phenomenon, say, the current packet being sent to a target networking interface `if0`. Assume that the packets are independent and come from the same background. For a sequence of n packets, the decision "route the current packet to `if0`" can be modeled by $B(p)$ and the number of packets which are actually routed to `if0` can be modeled by $B(n, p)$, the binomial distribution with parameters (n, p).

12.5.3. *Poisson distribution*

An r.v. X taking values in $\mathbb{N}$ has a *Poisson distribution* with parameter λ, noted by $P(\lambda)$ if

$$\forall k \in \mathbb{N}, \quad P(X = k) = e^{-\lambda} \frac{\lambda^k}{k!} \qquad [12.20]$$

We have:

$$E[X] = \lambda, \qquad Var[X] = \lambda. \qquad [12.21]$$

It can be shown that when we take a binomial distribution $B(n, p)$ and make n tending toward ∞ by keeping np constant, we get a Poisson distribution $P(\lambda)$ with $\lambda = np$. Recalling that the binomial distribution can be viewed as the sum of n independent Bernoulli outcomes. Thus, we can consider the Poisson distribution as a good approximation of the sum of a large number of independent and binary phenomena.

Many situations fulfill this condition, and quite a number of them are in telecommunications and networking. For instance, the number of people wanting to visit a target Website, the number of calls received by a hotline, the number of outgoing packets, etc. Thus, the Poisson distribution is one of the most fundamental distributions in telecommunications.

12.5.4. *Geometric distribution*

Consider again a sequence of independent r.v. having Bernoulli distribution. We are interested by X, the number of successive "0" before the first "1". X is an r.v. whose pmf are:

$$\forall k \in \mathbb{N} \;\; p_k = P(X = k) = q^k p \qquad [12.22]$$

This distribution is called *geometric distribution* with parameter p, and noted by $Geo(p)$. We have:

$$E[X] = \frac{q}{p}, \qquad Var[X] = \frac{q}{p^2}. \qquad [12.23]$$

The geometric distribution also has many applications. For instance, it can be used to model the number of re-transmissions of a packet within a connection-oriented communication (every packet must be successfully received) through a transmission channel with loss.

12.5.5. *Uniform distribution*

An r.v. X has a *uniform distribution* over the interval $[a, b]$, $a < b$, if its pdf, $f_X(x)$, is such that:

$$f_X(x) = \begin{cases} \frac{1}{b-a} & \text{for} \quad x \in [a, b] \\ 0 & \text{elsewhere} \end{cases} \qquad [12.24]$$

This distribution will be denoted as $U(a, b)$. We have:

$$E[X] = \frac{a+b}{2}, \qquad Var[X] = \frac{(b-a)^2}{12}.$$

A particular yet very useful case is the uniform distribution over $[0, 1]$, noted $U(0, 1)$. We can note that $f(x) = 1$ on $[0, 1]$ and is 0 elsewhere.

12.5.6. *Triangular distribution*

An r.v. X has a *triangular distribution* if its pdf first increases linearly until a certain value then decreases always linearly, forming in this way a triangle (see Figure 12.1). Mathematically, an r.v. X having a *triangular distribution*, noted by $T(a, b, c)$, is defined by three parameters $a < b < c$ as follows:

$$f_X(x) = \begin{cases} \frac{2(x-a)}{(b-a)(c-a)} & a \leq x \leq b \\ \frac{2(c-x)}{(c-b)(c-a)} & b \leq x \leq c \\ 0 & \text{elsewhere} \end{cases} \qquad [12.25]$$

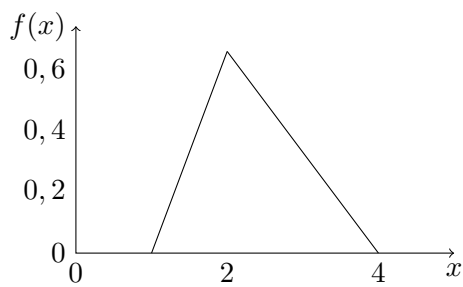

Figure 12.1. *pdf of the triangular distribution* $T(1, 2, 4)$

Its cumulative distribution function (CDF) is given by:

$$F_X(x) = \begin{cases} \frac{(x-a)^2}{(b-a)(c-a)} & a \leq x \leq b \\ \frac{b-a}{c-a} + \frac{(x-b)^2}{(c-b)(c-a)} & b \leq x \leq c \end{cases} \quad [12.26]$$

We have:

$$E[X] = \frac{a+b+c}{3}, \quad Var[X] = \frac{a(a-b)+c(c-a)+b(b-c)}{18}. \quad [12.27]$$

12.5.7. *Exponential distribution*

An r.v. X has an *exponential distribution* with parameter λ if its pdf $f_X(x)$ (see Figure 12.2) is given by:

$$f_X(x) = \begin{cases} \lambda e^{-\lambda x} & \text{for} \quad x > 0 \\ 0 & \text{elsewhere} \end{cases} \quad [12.28]$$

X is thus always positive. This distribution is denoted by $Exp(\lambda)$. We have:

$$E[X] = \frac{1}{\lambda}, \qquad Var[X] = \frac{1}{\lambda^2}.$$

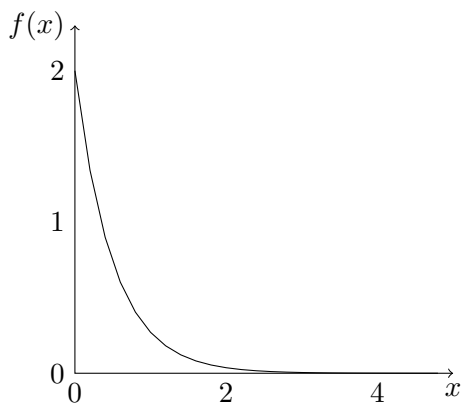

Figure 12.2. *pdf of the exponential distribution* $Exp(2)$

Its CDF is:

$$F(x) = 1 - e^{\lambda x} \qquad [12.29]$$

One of the most interesting properties of the exponential distribution is its *Markov* (or *memoryless*) property, which is:

$$\forall t > 0, \forall h > 0, \quad P(X > (t+h)/X > t) = P(X > h)$$

We see through this formula that, for an exponential distribution, the *past* ($X > t$) has no impact on the future evolution of X ($X > (t+h)$).

12.5.8. *Normal distribution*

An r.v. X has a *normal distribution* (or *Gauss distribution*) with parameters μ and $\sigma > 0$, noted by $N(\mu, \sigma^2)$, if its pdf is given by:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{(x-\mu)^2}{2\sigma^2}) \qquad [12.30]$$

Graphically, the function $f(x)$ (see Figure 12.3) looks like a "bell" which is symetric with respect to $x = \mu$ where $f(x)$ gets its maximum.

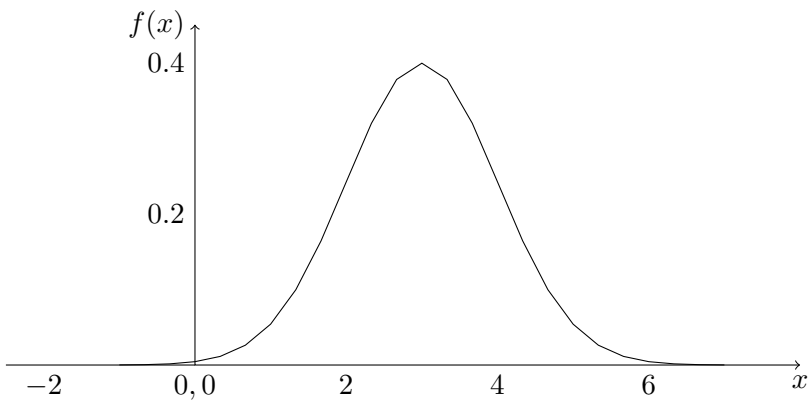

Figure 12.3. *pdf of the normal distribution* $N(3, 1)$

The CDF of $X \sim N(\mu, \sigma^2)$ is given by:

$$F(x) = \int_{t=-\infty}^{x} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}(\frac{t-\mu}{\sigma})^2\right) dt$$

The calculation of this integral is not obvious. Nevertheless, it suffices to calculate $N(0,1)$, called *standard normal distribution*, to be able to deduce the calculation for any $N(\mu, \sigma^2)$. Indeed, consider $X \sim N(0,1)$ whose pdf is given by:

$$f(x) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{x^2}{2})$$

and whose CDF, noted here by $\Phi(x)$, is given by:

$$\Phi(x) = \int_{t=-\infty}^{x} \frac{1}{\sqrt{2\pi}} \exp(-\frac{t^2}{2}) dt$$

Let $Y \sim N(\mu, \sigma^2)$, we have:

$$\begin{aligned} F_Y(x) &= \int_{t=-\infty}^{x} \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{1}{2}(\frac{t-\mu}{\sigma})^2) dt \\ &= \int_{t=-\infty}^{(x-\mu)/\sigma} \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{t^2}{2}) dt = \Phi(\frac{x-\mu}{\sigma}) \end{aligned}$$

Thus, $F_Y(x)$ can be deduced from $\Phi(x)$. The only calculation must be done is the one for $N(0,1)$. The latter is given under the form of table that we can find in handbooks.

The characteristic function of $X \sim N(\mu, \sigma^2)$ is given by:

$$\phi_X(u) = E[e^{iux}] = e^{i\mu u} e^{-\frac{1}{2}\sigma^2 u^2}$$

Since we have:

$$\frac{d^k \phi_X(x)}{dx^k}(u=0) = (i)^k E[X^k]$$

We obtain consequently:

$$E[X] = \mu, \qquad Var[X] = \sigma^2.$$

Thus, we have a physical interpretation of the two parameters of a normal distribution.

12.5.9. *Log-normal distribution*

A strictly positive r.v. X has a *log-normal* distribution if its logarithm has a normal distribution. The log-normal distribution, noted $LN(\mu, \sigma^2)$, is defined by two parameters μ and σ, such that $X \sim LN(\mu, \sigma^2)$ is equivalent to $\log X \sim N(\mu, \sigma^2)$. In other words, If we have an r.v. $X \sim N(\mu, \sigma^2)$, then $Y = e^X$ has a log-normal distribution $LN(\mu, \sigma^2)$. We have:

$$E[Y] = e^{\mu+(\sigma^2/2)}, \qquad Var[Y] = e^{2\mu+\sigma^2}\,(e^{\sigma^2} - 1)$$

One property of the log-normal distribution is that the proportion of high-end values is not negligible. From this point of view, the log-normal distribution has some similarity with the Pareto distribution (see next section) for the modeling of phenomena with "long tail".

12.5.10. *Pareto distribution*

The Pareto distribution is defined by a couple of positive parameters (s, α) where s gives the threshold from which the distribution is defined and α defines the slope of the function. The pdf (see Figure 12.4) of a Pareto distribution is given by:

$$f(x) = \frac{\alpha}{s}(\frac{x}{s})^{-(\alpha+1)} = \alpha s^{\alpha} x^{-(\alpha+1)},\ x \geq s, \qquad [12.31]$$

and its CDF:

$$F(x) = 1 - (\frac{x}{s})^{-\alpha},\ x \geq s. \qquad [12.32]$$

We have:

– $E[X] = \infty$ for $\alpha \leq 1$, $E[X] = \frac{\alpha s}{\alpha - 1}$ for $\alpha > 1$;

– $Var[X] = \infty$ for $\alpha \leq 2$, $Var[X] = \frac{\alpha s^2}{(\alpha-1)^2(\alpha-2)}$ for $\alpha > 2$.

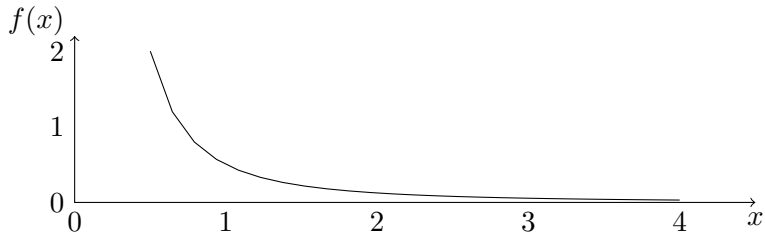

Figure 12.4. *pdf of the Pareto distribution with* $\alpha = 1$ *and* $s = 0.5$

Note that $P(X > x) = (\frac{x}{s})^{-\alpha}$ is a decreasing polynomial function. As a comparison, for the exponential distribution, we have $P(X > x) = e^{-\lambda x}$, i.e. the decrease is in exponential form. This means that the proportion of high-end values is much more important for a Pareto distribution than for an exponential one. This is the so-called *long tail* phenomenon. The Pareto distribution is more suitable for modeling quantities exhibiting this *long tail* phenomenon. One typical example is the distribution of the packet length in the Internet: there is actually a *non-negligible* part of *big* packets (sent by fulfilling the Ethernet MTU (Maximum Transmission Unit) of 1,500 bytes).

12.6. Joint probability distribution of multiple random variables

12.6.1. *Definition*

Let $X_i, i = 1, \ldots, n$ be a set of r.v. which are defined on the same sample space Ω. $X = (X_1, \ldots, X_n)$ form a *vector of random variables*. It is possible that $X = (X_1, \ldots, X_n)$ are *independent and identically distributed (i.i.d)* r.v. This is the case for a sequence of periodic (generally independent) measurements from a same system (same distribution).

In general cases, the $(X_1, \ldots, X_n)$ are not independent and may have different distributions. The distribution of X is the joint probability distribution of $\{X_i\}_{i=1,\ldots,n}$. For the sake of simplicity, we consider the case of two r.v. X, Y. The CDF of the joint probability distribution of $Z = (X, Y)$ is:

$$F_Z(x, y) = F_{X,Y}(x, y) = P(X \leq x, Y \leq y)$$

the pdf $f_{X,Y}(x,y)$ is given by:

$$f_{X,Y}(x,y) = \frac{\partial^2 F_{X,Y}(x,y)}{\partial x\, \partial y}$$

from which we deduce the *marginal* pdf of X (respectively Y), noted $f_X(x)$ (respectively $f_Y(x)$):

$$f_X(x) = \int_{y\in\mathbb{R}} f_{X,Y}(x,y)dy, \quad f_Y(y) = \int_{x\in\mathbb{R}} f_{X,Y}(x,y)dx$$

12.6.2. *Independence and covariance*

Concerning the independence between X and Y, we have the following equivalent criteria:

– X and Y are independent if and only if $f_{X,Y}(x,y) = f_X(x)f_Y(y)$;

– X and Y are independent if and only if $\phi_{X,Y}(u_x,u_y) = \phi_X(u_x)\phi_Y(u_y)$ where $\phi_{X,Y}$ (respectively ϕ_X, ϕ_Y) is the characteristic function of (X,Y) (respectively$X$, Y).

We introduce here a new concept which is the *covariance* of X and Y, noted $Cov(X,Y)$. It is defined by:

$$Cov[X,Y] = E[\,(X - E[X])(Y - E[Y])\,] \qquad [12.33]$$

After some calculus, we obtain another expression of $Cov[X,Y]$:

$$Cov[X,Y] = E[XY] - E[X]E[Y] \qquad [12.34]$$

12.6.3. *Mathematical expectation*

Here are some properties about mathematical expectation and the variance:

– $E[X+Y] = E[X] + E[Y]$ (X and Y are not necessary independent);

– more generally, for $a \in \mathbb{R}, b \in \mathbb{R}$, we have $E[aX + bY] = aE[X] + bE[Y]$;

– $Var[X+Y] = Var[X] + Var[Y] + 2Cov[X,Y]$;

– if X and Y are independent, then:

$$E[XY] = E[X]E[Y] \quad \text{and} \quad Var[X+Y] = Var[X] + Var[Y]$$

It is thus obvious that if X and Y are independent, $Cov[X,Y] = 0$. *Attention: the reciprocal is false.*

12.7. Some interesting inequalities

12.7.1. *Markov's inequality*

Let X be an r.v. such that $P(X < 0) = 0$ and $E[X] < \infty$, then for all $t > 0$:

$$P(X \geq t) \leq \frac{E[X]}{t}. \qquad [12.35]$$

This relation, called *Markov's inequality*, provides a simple, yet rather loose, way for seeking a bound for the probability of being beyond a certain threshold. For example, the probability that the occupancy of the buffer of a router goes beyond a threshold value. The bound obtained by the Markov inequality is often too conservative (too big), the Chebyshev inequality provides a more precise estimate.

12.7.2. *Chebyshev's inequality*

Let X be an r.v. with $E[X] < \infty$ and $Var[X] = \sigma^2 < \infty$, then for all $t > 0$:

$$P\Big(\big|X - E[X]\big| \geq t\Big) \leq \frac{\sigma^2}{t^2} \qquad [12.36]$$

which can also be put under the following form:

$$P\Big(\big|X - E[X]\big| \geq t\sigma\Big) \leq \frac{1}{t^2} \qquad [12.37]$$

This inequality is called *Chebyshev's inequality*.

12.7.3. *Cantelli's inequality*

Chebyshev's inequality deals with the *absolute* deviation from the mean. Sometimes, we wish to know *one-side* deviation. *Cantelli's inequality* can be used in this case.

Here is its canonical form. Let X be an r.v. with $E[X] = \mu < \infty$ and $Var[X] = \sigma^2 < \infty$, then:

$$P(X - \mu \leq \lambda) \leq \frac{\sigma^2}{\sigma^2 + \lambda^2} \qquad \text{if } \lambda < 0 \qquad [12.38]$$

$$P(X - \mu \leq \lambda) \geq 1 - \frac{\sigma^2}{\sigma^2 + \lambda^2} \qquad \text{if } \lambda > 0 \qquad [12.39]$$

By denoting $\lambda = v - \mu$, we can rewrite Cantelli's inequality as:

$$P(X \leq v) \leq \frac{\sigma^2}{\sigma^2 + (v - \mu)^2} \qquad \text{if } v < \mu \qquad [12.40]$$

$$P(X > v) \leq \frac{\sigma^2}{\sigma^2 + (v - \mu)^2} \qquad \text{if } v > \mu \qquad [12.41]$$

These expressions gives *explicitly* the probability of being *lower* (respectively *higher*) than a given threshold value v on the *left* (respectively *right*) side of the mean. This inequality is also thus called the *one-sided inequality*.

12.8. Convergences

We consider a sequence of r.v. X_n, $n \in \mathbb{N}$, as well as a single r.v. X. The $\{X_n\}_{n \in \mathbb{N}}$ and X are defined on the same space $(\Omega, \mathcal{F}, P)$.

We are interested by the case where the sequence $\{X_n\}_{n \in \mathbb{N}}$ converges to X when n increases. In this case, we are also interested in the way in which this convergence is made. In the remainder of the chapter, we present four types of convergence.

12.8.1. *Types of convergence*

12.8.1.1. *Convergence in mean*

We suppose that $E[X_n]$ exists as well as $E[X]$. In this case, by definition, X_n converges *in mean* toward X if:

$$\lim_{n\to\infty} E[(X_n - X)] = 0$$

In a similar way, assume that $E[X_n^2]$ and $E[X^2]$ exist. By definition, X_n converges *in mean square* (MS) toward X if: s

$$\lim_{n\to\infty} E[(X_n - X)^2] = 0$$

Generally speaking, if X_n and X are in $\mathcal{L}^k(\Omega, \mathcal{F}, P)$ (i.e. $E[X_n^k]$ and $E[X^k]$ exist), then by definition, X_n converges *in the k-th mean* toward X if:

$$\lim_{n\to\infty} E[(X_n - X)^k] = 0$$

12.8.1.2. *Convergence in probability*

By definition, X_n converges *in probability* toward X if:

$$\forall \epsilon > 0, \forall \delta > 0, \exists N, \text{ such that } \forall n > N, P(|X_n - X| > \delta) \leq \epsilon$$

12.8.1.3. *Almost sure convergence*

12.8.1.3.1. Definition

Let us consider a particular trial of the sequence $\{X_n\}$ and X, i.e. we take some $\omega \in \Omega$ and consider the (deterministic) sequence $\{X_n(\omega)\}$ and the (deterministic) value $X(w)$. It is possible that we do not have $\lim_{n\to\infty} X_n(\omega) \to X(w)$, indeed $X_n(\omega)$ and $X(w)$ are particular outcomes.

The *almost sure convergence* (A.S.), if it exists, ensures that the probability of having $\lim_{n\to\infty} X_n(\omega) \to X(w)$ is 1 for almost all choices of ω. By definition, X_n almost surely converges to X if and only if there is a negligible event E such that:

$$\omega \in \overline{E} \Longrightarrow \lim_{n\to\infty} X_n(\omega) = X(\omega)$$

12.8.1.3.2. Borel–Cantelli lemma

We consider $\{A_n\}_{n\in\mathbb{N}}$ a sequence of subsets of a set Ω. For each n, we define the subset B_n (also noted as $\sup_{i\geq n} A_i$):

$$B_n = \sup_{i\geq n} A_i = \cup_{i=n}^{\infty} A_i$$

Thus, $B_{n+1} \subset B_n$, when n increases, B_n decreases till the limit which is $\cap_{n=1}^{\infty} B_n$. The latter is, by definition, the *upper limit* of A_n, noted $\limsup_n A_n$:

$$\limsup_n A_n = \lim_{n\to\infty} B_n = \cap_{n=1}^{\infty} \cup_{i=n}^{\infty} A_i$$

from where, we have the following result. Consider an outcome $\omega \in \Omega$, the fact that $\omega \in \limsup_n A_n$ is equivalent to the fact that there is an infinity of j such that $\omega \in A_j$. The lemma of Borel–Cantelli lemma goes in this direction.

Borel–Cantelli lemma: in the probability space $(\Omega, \mathcal{F}, P)$, we consider $\{A_n\}$, $n \geq 1$, a sequence of events which are elements of $\mathcal{F}$. We have:

$$\sum_{n=1}^{\infty} P(A_n) < \infty \Longrightarrow P(\limsup_n A_n) = 0$$

The Borel–Cantelli lemma provides a useful criterion to determine the almost sure convergence of a sequence of r.v. X_n toward an r.v. X. Indeed, the almost sure convergence of X_n toward X admits the following criterion:

$$\forall \epsilon > 0, P(\limsup_n |X_n - X| \geq \epsilon) = 0$$

12.8.1.4. *Convergence in distribution*

A sequence X_n of r.v. is said to *converge in distribution* toward an r.v. X if:

$$\lim_{n\to\infty} P(X_n \leq x) = P(X \leq x)$$

for each x at which $P(X \leq x)$ is continuous. We also use the term *converge in law*.

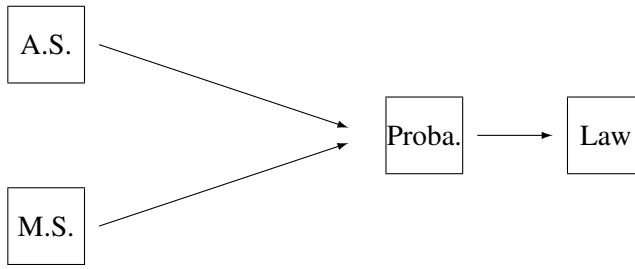

Figure 12.5. *Relations between convergences*

12.8.1.5. *Relations between the convergences*

Some convergences are "stronger" than others in the sense that the former implies the latter; some of these relations (see also Figure 12.5) are as follows:

– the almost sure convergence (A.S.) implies the convergence in probability, the latter implies the existence of a sub-sequence which almost surely converges;

– the convergence in mean square (M.S.) implies the convergence in probability;

– the convergence in the k-th mean (L^k) implies the convergence in the $(k-1)$-th mean. In particular, convergence in M.S. implies that in mean;

– the convergence in M.S. implies existence of a sub-sequence which almost surely converges;

– the convergence in probability implies the convergence in distribution. The latter is the "weakest" convergence.

12.8.2. *Law of large numbers*

We consider a sequence $\{X_n\}_{n\in\mathbb{N}}$ of r.v. The (strong) *law of large numbers* stipulates that if the X_n are i.i.d. and their mean value $E[X_n] = m$ exists, then the sequence $\{Y_n\}_{n\in\mathbb{N}}$ defined by:

$$Y_n = \frac{\sum_{i=1}^{n} X_i}{n}$$

almost surely converges to m. There is also a *weak law of large numbers* which only stipulates a convergence in mean.

In the case where $\{X_n\}$ are in $\mathcal{L}^2$ ($E[X_n^2]$ exist), there is almost sure convergence even if X_n are not independent but simply *non-correlated* (i.e. $\{X_n\}$ are such that $Cov[X_i, X_j] = 0$ for $i \neq j$).

12.8.3. *Central limit theorem*

Consider a sequence $\{X_i\}_{i=1,\ldots,n}$ of i.i.d. r.v., which is in $\mathcal{L}^2$, with $E[X_i] = m$ and $Var[X_i] = \sigma^2$. We define:

$$Y_n = \frac{\sum_{i=1}^n X_i}{n}.$$

The (strong) law of large numbers states that Y_n converge almost surely to m. We will see later that Y_n, which is termed as *sample mean*, is an estimator of m from the "sample" $\{X_i\}_{i=1,\ldots,n}$, the latter represents data taken from n independent observations of a same system.

The problem here is to know the distribution of the difference between Y_n and m. The *Central Limit Theorem* states that the r.v. Z_n given by:

$$Z_n = \frac{Y_n - m}{\sigma/\sqrt{n}} = \frac{\sum_{i=1}^n (X_i - m)}{\sigma\sqrt{n}},$$

which is the "normalized" difference (whose variance is 1) related to the mean (m), converges in distribution to $N(0, 1)$, which is:

$$\lim_{n\to\infty} P\Big(\frac{\sum_{i=1}^n (X_i - m)}{\sigma\sqrt{n}} \leq x\Big) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp(-\frac{y^2}{2}) dy$$

This result has a great importance in practice. In particular, it will enable us to calculate the *confidence interval* (see section 13.3.6).

13

An Introduction to Statistics

In this chapter, we present, in a very summary way, some rudimentary notions of statistics, in particular on parameters estimation and hypothesis testing. This chapter is primarily a reminder of the concepts which are used in a recurring way in this book, implicitly or explicitly. For an in-depth study, it is recommended to consult specialized books, for example [WAS 04, MOO 09, DEG 81], francophone readers can also consult [LEJ 11].

13.1. Introduction

The statistics can be viewed, to some extent, as a complement to the theory of probability. Indeed, the latter attempts to study various types of stochastic behavior so that we can make predictions. The purpose of statistics is to identify the characteristics of a stochastic process from the data generated by this one, and, idealistically, identify the underlying probability distribution.

The study of a stochastic process starts usually with the observation of its behavior through measurements. We thus obtain, after each observation campaign, a sequence of N outcomes. It is said that we have a *sample* of size N (or N-sample). Let us take the example of a router for which we want to study the buffer's occupancy. An observation[1] over 9 sec, at a rate of one observation per second, gives us the sequence (denoted as $\mathcal{S}$) which is given by Table 13.1 and is illustrated by Figure 13.1.

1 We voluntarily limit the observations to 9 for the sake of simplicity. In the real world, an observation campaign usually produces a much larger sized sample.

Performance Evaluation by Simulation and Analysis with Applications to Computer Networks, First Edition. Ken Chen. © ISTE Ltd 2015. Published by ISTE Ltd and John Wiley & Sons, Inc.

Second	1	2	3	4	5	6	7	8	9
Packets number	42	67	100	89	56	23	0	5	34

Table 13.1. *Outcomes of the observation $\mathcal{S}$*

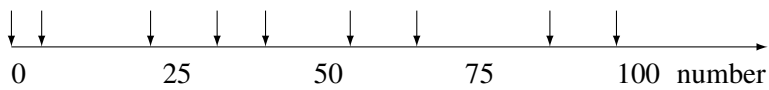

Figure 13.1. *Location of data in the sample $\mathcal{S}$*

There are two possible visions on a sample, according to whether we are interested by their nature or by the observed values. Indeed:

– a sample contains data which are intrinsically *unknown in advance*, these data are thus random variables (r.v.) that we will denote as $(X_1, X_2, \ldots, X_N)$;

– a given sample is constituted by concrete data coming from experiments, these data are deterministic values that we will denote as $(x_1, x_2, \ldots, x_N)$.

We will always have to deal with this duality. By convention, for an outcome of a sample, say the ith, X_i denotes the r.v., and x_i gives the observed value. Unless specified otherwise, X_i are independent and identically distributed (i.i.d.) r.v.

13.2. Description of a sample

It is useful to describe a sample by giving its characteristics. We would like to point out that the description presented here is *deterministic* in the sense that it concerns a given sample with the observed values $\{x_i\}$ and does not extend beyond it. We will be interested later in estimations from samples considered as constituted by r.v. X_i.

13.2.1. *Graphic representation*

Graphics often provide very rich information. The graphics which most directly provide an idea on the distribution should be its *histogram*, which

can be viewed as a kind of empirical probability density function (pdf), or, alternately, its empirical cumulative distribution function (CDF) (see section 13.2.6). Concerning the graphical form, it often depends on practices and/or application domain. For example, in business world, we often use *pie chart*.

13.2.2. *Mean and variance of a given sample*

The *mean* of a given sample $\mathcal{S} = \{x_i\}_{i=1,\ldots,N}$ related to an r.v. X is defined as the sum of the values divided by their number:

$$\overline{X} = \frac{\sum_{i=1}^{N} x_i}{N}$$

The *variance* of this given sample is defined as:

$$\sigma^2 = \frac{\sum_{i=1}^{N} (X_i - \overline{X})^2}{N}$$

It is to be pointed out that these calculations concern a set of *deterministic* values. In order to describe the behavior of a set of random values, we will introduce later formulae for *sample mean* and *sample variance*.

13.2.3. *Median*

The *median* of a sample $\mathcal{S}$ is the value located at the middle of all of the (ordered) values in $\mathcal{S}$. In other words, the median splits the values of $\mathcal{S}$ (except itself) into two subsets having the same number of values, in one of them, the values are greater than the median, and in the other one, they are smaller than the median. In our example, $\mathcal{S} = \{0, 5, 23, 34, 42, 56, 67, 89, 100\}$, the median is 42 and the two subsets are $\{0, 5, 23, 34\}$, $\{56, 67, 89, 100\}$.

In cases where $\mathcal{S}$ has an even number of values, the median will be, by convention, the mean value of the two values "in the middle" among the sequence of ordered values of $\mathcal{S}$. In this case, the median will be a *fictive* value since it is not a value of $\mathcal{S}$, whereas it always splits $\mathcal{S}$ into the two subsets as described previously. A simplest example follows: for $\mathcal{S} = \{2, 4\}$, the median will be 3 which splits $\mathcal{S} = \{2, 4\}$ in to $\{2\}$ and $\{4\}$. Taking again our example by adding 50 to it, the new median will be 46 which is the average between 42 and 50.

13.2.4. *Extremities and quartiles*

The extremes values, i.e. the minimal value (in our example $\min(\mathcal{S}) = 0$) and the maximal value (in our example $\max(\mathcal{S}) = 100$), give the dynamic range of the sample.

Two other parameters of interest are the first and the third *quartile*, denoted, respectively, by Q_1 and Q_3. By definition, over an ordered sample, $[\min, Q_1]$ and $[Q_3, \max]$ are subsets containing each 25% of values, the subset $[\min, Q_1]$ being the lowest quarter and $[Q_3, \max]$ the highest one. There is also the *interquartile range* (IQR) defined as $Q_3 - Q_1$, which thus represents the central zone around the median with 50% of the values.

As the median divides the whole set of ordered values into two subsets having the same numbers, it can be considered as Q_2. One method[2] that we can use to calculate Q_1 and Q_3 consists of considering them as the "median" of the respective subsets.

These five values, namely <min, Q_1, median, Q_3, max>, provide a rather precise description of the dispersion of the sample. In our case, we have $<0, 14, 42, 78, 100>$; $Q_1 = 14$ and $Q_3 = 78$ are not in $\mathcal{S}$.

13.2.5. *Mode and symmetry*

The *mode* is the most probable value (that having the highest frequency). On a histogram, the mode is thus the value which is under the peak. Formally speaking, the mode of a continuous r.v. is x such that the pdf $f(x)$ gets its maximum; for a discrete r.v., it is x_i such that the probability mass function (pmf) $f(x_i)$ is the greatest. If the pdf or pmf are not monotonically increasing before (respectively, decreasing after) x, but has several local maxima, the distribution is said to be *multimodal*; otherwise, it is *unimodal.*

On the basis of this parameter, we can see whether the distribution is symmetrical or not related to its mode.

---

2 Using this method, the median itself is not included in the subsequent calculation of Q_1 and Q_3. Other methods exist which include the median.

13.2.6. *Empirical cumulative distribution function and histogram*

13.2.6.1. *Empirical cumulative distribution function*

From a sample $\mathcal{S} = \{x_i\}_{i=1...n}$ of an r.v. X, we can define its *empirical CDF* related to the sample $\mathcal{S}$. This empirical CDF, denoted by F_n, is defined as follows:

$$\forall x \in \mathbb{R}, F_n(x) = \frac{\sum_{i=1}^{n} I(x, x_i)}{n} \quad [13.1]$$

where $I(x, x_i)$ is the indicator function defined by:

$$\forall x < x_i, I(x, x_i) = 0, \quad \forall x \geq x_i, I(x, x_i) = 1$$

In other words, the function $F_n(x)$ gives, for each x, the proportion of the observed values which are lower than it. Graphically (see Figure 13.2), it is a *step function* with unit increase at each observed value x_i. The function $F_n(x)$ constitutes an approximation of the CDF of X, thus the term *empirical CDF*.

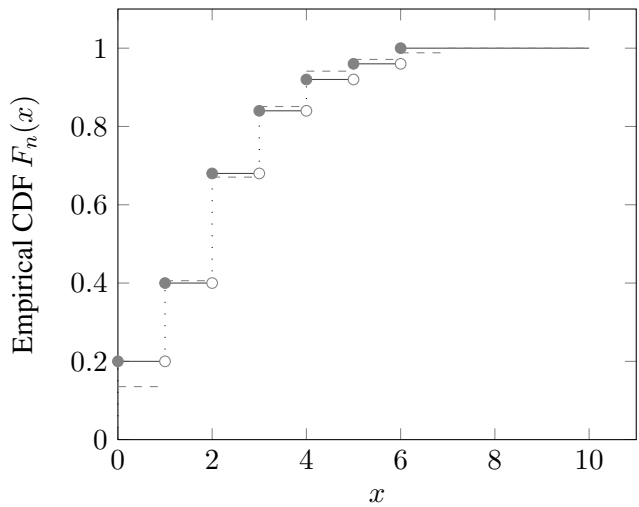

Figure 13.2. *An example of empirical CDF: the empirical CDF is in solid lines. The real CDF is reported in dashed lines, whereas the vertical loosely dotted lines mark the jumps. There is a pretty good fit between the two CDF*

13.2.6.2. *Histogram*

From a sample $\mathcal{S} = \{x_i\}_{i=1...n}$ of an r.v. X, we can also get an *histogram* in order to get an empirical representation of its distribution. The term *histogram* is devoted to *continuous* r.v., it represents graphically an approximation of the pdf of the r.v.

For discrete (categorical) r.v., the construction method is almost the same, but it is called *bar chart* (or *bar graph*) due to the discontinuity of the values (see below).

Roughly speaking, we obtain a histogram with the following process:

– Divide $\mathbb{R}$ into K contiguous intervals, $I_i, i = 1, \ldots, K$, $\mathbb{R} = \cup_{i=1,\ldots,K} I_i$. The intervals $\{I_i\}_{i=1,\ldots,K}$ do not necessary have the same width.

– Count for each interval I_i the number of times, denoted as n_i, where one of the values of $\mathcal{S}$ falls into I_i.

– Draw over each I_i a rectangle whose height is equal to n_i/N.

We obtain in this way the histogram of X from $\mathcal{S}$.

In the case of a discrete r.v., say, for the sake of simplicity, an r.v. X with K values $x_i, i = 1, \ldots, K$, the $\{I_i\}_{i=1,\ldots,K}$ are reduced to the single values $\{x_i\}, i = 1, \ldots, K$.

A more formal description of the construction of a histogram is given below:

– We define at first the $K - 1$ cutting points $\{c_i\}_{i=1...(K-1)}$ with $-\infty < c_1 < c_2 < \cdots < c_{K-1} < \infty$, to form K intervals as follows:

- $I_1 = (-\infty, c_1]$;

- $I_K = (c_{K-1}, \infty)$;

- if $K > 2$, I_k for $2 < k \leq K - 1$ is given by $I_k = [c_k, c_{k+1})$;

– f_k, the frequency related to I_k, is given by:

$$f_k = \frac{\text{Card}\{x_i \in I_k / i = 1 \ldots, n\}}{n} \qquad [13.2]$$

We can see that the histogram of an r.v. (continuous) yields an approximation of its pdf, whereas the bar chart of an r.v. (discrete) yields the one of its pmf. Figure 13.3 gives an example of a histogram which is actually a *bar chart* since the underlying r.v. is discrete (categorial). It is obtained with the same data as those of Figure 13.2, both are produced with the data of section 13.4.2.2 which have also been summarized in Table 13.3.

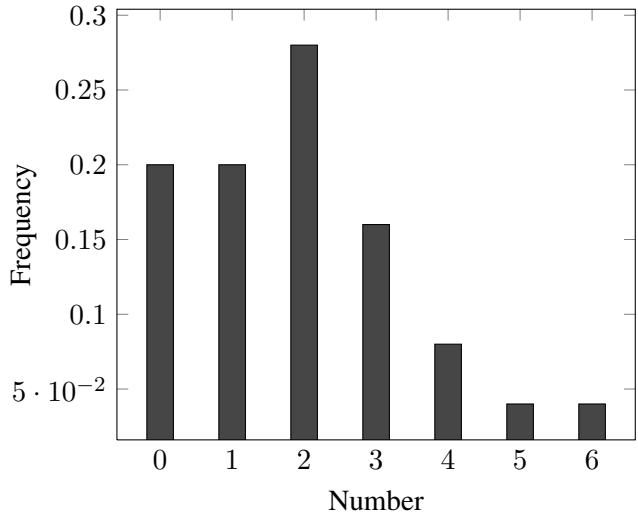

Figure 13.3. *A histogram/bar chart*

There is no rule for the choice of the intervals either on their number or on the size of each of them. The first concern consists of making sure that there is a significant number of observations in each interval. Cutting must also have a sufficient level of granularity to be able to capture specificity of the underlying, and unknown, distribution of X. Generally, we take intervals of equal width and roughly $\sqrt{N}$ intervals for a sample of size N.

13.2.6.3. *Conclusion*

In conclusion, from a sample, we can define:

– an empirical CDF which is an approximation of the true CDF, the latter being uniquely defined;

– a histogram (respectively, bar chart) which is an approximation of the pdf (respectively, pmf) of the r.v. in the case of a continuous (respectively, discrete)

r.v. There is no unique form for the histogram of an r.v., it may take more or less different forms depending on the cutting.

13.3. Parameters estimation

13.3.1. *Position of the problem*

Very often, we have to deal with a random quantity X for which we do not know its distribution $\mathcal{L}$. Through independent experiments, we can obtain outcomes of X. As these values are not foreseeable, we have, at the end of n experiment, a sequence of n i.i.d. r.v., $\mathcal{S} = \{X_i\}_{i=1,\ldots,n}$. The latter constitutes a *sample* of X, as already introduced in (see section 13.1).

The objective of an estimation consists of proposing an identification, which is the most faithful possible (according to certain criteria), of some parameters (such as its mean and variance) of the distribution $\mathcal{L}$, or, idealistically, the underlying distribution itself.

13.3.2. *Estimators*

13.3.2.1. *Definition*

We are interested in θ, a parameter of $\mathcal{L}$ which we want to estimate. Examples of θ are the mean and the variance. An *estimator* of θ, denoted as $\hat{\theta}$, is a function of the sample $\mathcal{S} = \{X_i\}_{i=1,\ldots,n}$.

13.3.2.2. *Quality of an estimator*

An estimator $\hat{\theta}$ is itself an r.v. whose value depends on those of the outcomes $(X_1, \ldots, X_n)$. We can assess the *quality* of an estimator through some criteria. Here are some main qualities:

– *biased versus unbiased*: if $E[\hat{\theta}] - \theta \neq 0$, this means that the estimator $\hat{\theta}$ will introduce a systematic error, it is said to be *biased*. An *unbiased* estimator does not commit this systematic error. $\overline{X}$ and S^2 (see next section) are examples of *unbiased* estimators. If, in the expression of S^2, we replaced the $n-1$ of denominator by n, there would be a systematic error;

– *minimum variance*: the smaller the variance of an estimator ($Var[\hat{\theta}]$), the more faithful its estimation will be;

– *consistency*: an estimator $\hat{\theta}$ is said to be *consistent* if it converges toward θ in probability, i.e.:

$$\forall \epsilon > 0, \lim_{n \to \infty} P(|\hat{\theta} - \theta| < \epsilon) = 1$$

13.3.3. *Sample mean and sample variance*

13.3.3.1. *Definition*

Hereafter, we give two estimators, respectively, for the mean and the variance, which are among the most commonly used estimators. The former is called *sample mean* and is denoted by $\overline{X}$, the latter is called *sample variance* and is denoted by S^2:

$$\overline{X} = \frac{\sum_{i=1}^{n} X_i}{n} \quad [13.3]$$

$$S^2 = \frac{\sum_{i=1}^{n} (X_i - \overline{X})^2}{n-1} \quad [13.4]$$

13.3.3.2. *Properties of $\overline{X}$ and S^2*

For an r.v. X with $E[X] = m < \infty$ and $Var[X] = \sigma^2 < \infty$, we have the following results for the two estimators $\overline{X}$ and S^2:

– $\overline{X}$ is an unbiased and consistent estimator of m;

– S^2 is an unbiased and consistent estimator of σ^2;

– if $X \sim N(\mu, \sigma^2)$, $\overline{X}$ and S^2 are in addition of *minimum variance*;

– if X has Poisson distribution or Bernoulli distribution, $\overline{X}$ is in addition of *minimum variance*.

13.3.4. *Maximum-likelihood estimation*

The method called *maximum-likelihood estimation* (MLE) provides a generic way to estimate one or more parameters. The principle of MLE consists of providing an estimate which makes maximum the probability of getting the *observed* values from the sample. It has been proven that the MLE is consistent and asymptotically normal.

Let $(x_1, \ldots, x_n)$ be the numerical values of the sample. The *likelihood function*, $l(\theta)$, is the joint probability of having the n values $(x_1, \ldots, x_n)$ and $l(\theta)$ is thus a function of $(x_1, \ldots, x_n)$. Depending on whether X_i is continuous or discrete, we express $l(\theta)$ through the pdf or pmf:

$$\begin{cases} l(\theta) = \Pi_{i=1}^{n} p(x_i) = h(x_1, \ldots, x_n, \theta), \ X \text{ discrete} \\ l(\theta) = \Pi_{i=1}^{n} f(x_i) = h(x_1, \ldots, x_n, \theta), \ X \text{ continuous} \end{cases}$$

The estimator of θ is the value $\hat{\theta}$ at which $l(\theta)$ is maximal. $\hat{\theta}$ is thus the solution of the following equation:

$$\frac{\partial l(\theta)}{\partial \theta} = 0$$

Often, it is more convenient to take $L(\theta) = \log(f(\theta))$, and the equation to be solved becomes:

$$\frac{\partial L(\theta)}{\partial \theta} = 0$$

This method can be generalized to estimate multiple parameters $(\theta_1, \ldots, \theta_k)$. In this case, it is necessary to solve the equations system:

$$\frac{\partial l(\theta_1)}{\partial \theta_1} = 0, \ldots, \frac{\partial l(\theta_k)}{\partial \theta_k} = 0.$$

EXAMPLE 13.1.– We consider here the case of the Pareto distribution (see section 12.5.10). The pdf of the latter is given by:

$$f(x) = \frac{\alpha}{s}(\frac{x}{s})^{-(\alpha+1)} = \alpha s^{\alpha} x^{-(\alpha+1)}, \ x \geq s,$$

Let $(x_1, \ldots, x_n)$ be our sample. As s cannot be, by its definition, bigger than any actually observed value, we take the maximum value authorized by the sample, i.e. $\hat{s} = \min\{x_i\}$. We now have to estimate α. Using the MLE, i.e. we seek to maximize:

$$l(\alpha) = \Pi_{i=1}^{n} \left(\alpha \hat{s}^{\alpha} x^{-(\alpha+1)} \right)$$

or, by using $L(\alpha) = \log l(\alpha)$:

$$L(\alpha) = n(\log \alpha + \alpha \log \hat{s}) - (\alpha + 1) \sum_{i=1}^{n} \log x_i$$

This leads to:

$$\frac{n}{\hat{\alpha}} + n \log \hat{s} - \sum_{i=1}^{n} \log x_i = 0$$

which gives:

$$\hat{\alpha} = \frac{n}{\sum_{i=1}^{n} \log x_i - n \log \hat{s}} = \frac{n}{\sum_{i=1}^{n} \log(x_i/\hat{s})}$$

13.3.5. *Method of moments*

The *method of moments* proposes an alternative approach to estimate the parameters of a distribution.

Let us assume that we are seeking to estimate a set of k parameters $(\theta_1, \ldots, \theta_k)$ of a certain distribution $\mathcal{L}$ from a sample of n outcomes $\mathcal{S} = (X_1, \ldots, X_n)$. We can calculate the k first moments, $m_j, 1 \leq j \leq k$, of $\mathcal{L}$ as a function of the parameters $(\theta_1, \ldots, \theta_k)$: $m_j(\hat{\theta_1}, \ldots, \hat{\theta_k})$. On the other hand, we can calculate the *sample jth moment*, $j = 1, \ldots, k$, with:

$$\overline{X^j} = \hat{m_j} = \frac{1}{n}(\sum_{i=1}^{n} X_i^j), \quad 1 \leq j \leq k$$

The method of moments consists of associating each m_j with its estimator $\overline{X^j}$. This leads to a system of k equations with k values to be determined which are the k parameters $(\theta_1, \ldots, \theta_k)$.

EXAMPLE 13.2.– Consider the Poisson distribution $P(\lambda)$ which is determined by a single parameter λ. The latter is nothing but the mean value (first moment m_1). According to the method of moments, we have:

$$\hat{\lambda} = \frac{1}{n}(\sum_{i=1}^{n} X_i) \qquad [13.5]$$

We recognize the classical estimator of the sample mean.

Let us take the normal distribution $N(\mu, \sigma^2)$. We have $m_1 = \mu$ and $m_2 = \mu^2 + \sigma^2$. Thus:

$$\begin{cases} \hat{\mu} = \frac{1}{n}\left(\sum_{i=1}^{n} X_i\right) \\ (\hat{\mu})^2 + (\hat{\sigma})^2 = \frac{1}{n}\left(\sum_{i=1}^{n} X_i^2\right) \end{cases}$$

We recognize the sample mean estimator for $\hat{\mu}$. On the contrary, for the estimator of variance ($\hat{\sigma}^2$), we have:

$$(\hat{\sigma})^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \hat{\mu})^2$$

This estimator is not the sample variance estimator, and it is a *biased* one.

13.3.6. *Confidence interval*

The *confidence interval* is a very important concept on statistics. It offers an effective way to assess the quality of an estimation. It is a fundamental tool used by simulations to indicate the quality of the results.

Consider the sample mean estimator $\overline{X}$. It is itself an r.v., with $E[\overline{X}] = E[X]$. If, moreover, $Var[X] = \sigma^2$ exists, the central limit theorem (see section 12.8.3) states that for large n, the r.v. Y with

$$Y = \frac{\overline{X} - E[X]}{\sigma/\sqrt{n}}$$

tends to the standard normal distribution $N(0, 1)$. Under these conditions, the confidence interval of the estimate of $E[X]$ with $100(1-\alpha)\%$ confidence is given by $[\overline{X} - E, \overline{X} + E]$, with $E = z(n-1, \alpha/2)\sigma/\sqrt{n}$ where $z(n-1, \alpha/2)$ is a coefficient such that:

$$P\left(-z(n-1, \alpha/2)) < \frac{\overline{X} - E[X]}{\sigma/\sqrt{n}} < z(n-1, \alpha/2)\right) = 1 - \alpha$$

In practice, we generally choose $\alpha = 0.05$ (confidence at 95%), which corresponds to $z(n-1, 0.025) = 1.96$ for $n > 30$. We use the value provided by S^2, which is a good *estimator* of σ^2, to complete the calculation of E.

13.4. Hypothesis testing

13.4.1. *Introduction*

Let us recall that we manage to "guess", as faithfully as possible, the "true" behavior of a stochastic quantity through partial knowledge given by some sample values. In the previous section, we presented some methods allowing us to identify a certain number of parameters of the stochastic quantity.

Here, we are interested in the agreement between the observed data and a distribution which we suppose to be that of the observed data. We speak about *hypothesis testing*. In this section, we do not aim to give a systematic presentation of *hypothesis testing*. We simply aim to present two methods among the most used in simulation studies, namely the Chi-square (χ^2) test and the Kolmogorov–Smirnov test (K-S test), for the test of *goodness of fit* between a sample data and a given distribution.

13.4.2. *Chi-square (χ^2) test*

13.4.2.1. *Principle*

Let $(x_1, \ldots, x_N)$ be a sample obtained from N observations of an r.v. X with unknown distribution. We make the hypothesis that this unknown distribution is $F(.)$. The *Chi-square* (χ^2) test proposes a method to check the degree of confidence of this assumption. Here is the algorithm:

1) Partitioner into K contiguous blocks, denoted as B_i, $i = 1, \ldots K$, the whole interval on which X is defined. The intervals B_i do not necessarily have the same width. Count the number of observations in each block, that of block B_i will be denoted by O_i:

$$O_i = \mathrm{Card}\{x_j / x_j \in B_i\}$$

2) Calculate the prediction by the assumed distribution $F(.)$ of the number of observations that should be in block B_i, denoted by E_i:

$$E_i = P[X \in B_i] \times N$$

3) Calculate the χ^2 indicator given by:

$$\chi^2 = \sum_{i=1}^{K} \frac{(O_i - E_i)^2}{E_i}$$

4) Determine the degree of freedom $d = K - 1 - l$ where l is the number of parameters of $F(.)$ which have been estimated from the sample $(x_1, \ldots, x_N)$. For example, if the assumed distribution is Poisson, the latter has one parameter λ, if λ has itself been estimated (by using, for instance, equation [13.5]), in this case $l = 1$.

5) Consult the table of χ^2 with the good values of d and α to obtain the threshold value χ^2_α for a confidence level of $1 - \alpha$. A set of several useful values of thresholds is provided in Table 13.2.

Degree of freedom (d)	2	3	4	5	6	7	8	9	10
Threshold χ^2_α ($\alpha = 5\%$)	0.1	0.58	1.06	1.46	2.2	2.83	3.49	4.17	4.87

Table 13.2. *Some useful threshold values for the χ^2 test*

6) If $\chi^2 < \chi^2_\alpha$, then the hypothesis is accepted with a confidence of $1 - \alpha$ (the most commonly used value is $\alpha = 0.05$); otherwise, the hypothesis has to be rejected.

If the test fails, we may proceed to a new organization of the blocks. It is clear that we may want to avoid having small E_i. A practical rule consists of establishing the blocks so that $E_i \geq 5$. If a first partition does not lead to this situation, we proceed by regrouping some blocks.

13.4.2.2. *Example*

We give below an example to illustrate the use of the χ^2 test. Consider a discrete r.v. (actually the number of visitors to a Website) from which we have obtained the following measurements:

```
2, 2, 4, 2, 4, 2, 2, 5, 2, 6, 1, 4, 1, 0, 0, 2, 1, 2, 0, 1,
3, 3, 1, 3, 1, 0, 0, 2, 0, 3, 0, 0, 2, 0, 3, 6, 2, 0, 3, 1,
1, 2, 4, 2, 1, 5, 2, 1, 3, 3
```

From this sample, we can draw a *bar chart* (see Figure 13.3) which corresponds to the first lines of Table 13.3. This suggests the assumption of a Poisson distribution. Prior to calculating the predicted numbers E_i by this Poisson distribution, we must estimate its parameter λ, which is nothing but its mean. We have:

$$\hat{\lambda} = \frac{10 + 2 \times 14 + 3 \times 8 + 4 \times 4 + 5 \times 2 + 6 \times 2}{50} = \frac{100}{50} = 2.$$

We can then calculate the predicted occurrences (E_i, $i = 0, \ldots, 6$) over a total of $N = 50$ outcomes:

$$E_i = N \times P[X = i] = N \times \exp(\hat{\lambda}) \frac{\hat{\lambda}^i}{i!}$$

These values are also shown in Table 13.3.

Values i	0	1	2	3	4	5	6	≥ 7
Occurrences (O_i)	10	10	14	8	4	2	2	0
Frequency ($O_i/50$)	0.2	0.2	0.28	0.16	0.08	0.04	0.04	0
Probability ($P(X = i)$)	0.1354	0.2706	0.2706	0.1804	0.0902	0.036	0.012	0.0046
Estimated (E_i)	6.77	13.53	13.53	9.02	4.51	1.80	0.60	0.23

Table 13.3. *Chi-square (χ^2) test: first partition*

By noting that the last three blocks are too weak in terms of occurrences, we gather them together, which leads to Table 13.4.

Values i	0	1	2	3	4	≥ 5
Occurrences (O_i^g)	10	10	14	8	4	4
Estimated (E_i^g)	6.77	13.53	13.53	9.02	4.51	3.63

Table 13.4. *Chi-square (χ^2) test: second partition*

The χ^2 indicator is obtained with:

$$\chi^2 = \sum_{i=0}^{5} \frac{(O_i^g - E_i^g)^2}{E_i^g} = 0.75$$

The degree of freedom is $6-1-1=4$. By consulting Table 13.2, we check readily that $\chi^2 = 0.75 < \chi^2_\alpha = 1.06$, the hypothesis of a Poisson distribution is acceptable at 95% confidence level. As an anecdote, the sequence of the 50 values was actually obtained with a pseudorandom number generator (PRNG) according to a Poisson distribution of parameter $\lambda = 2$.

It should be noted that this sample of 50 measurements is not completely representative. In addition, partitioning is a delicate operation as illustrated by the following example. If we carry out the cutting given by Table 13.5, i.e. by gathering the last four blocks instead of the last three, the χ^2 indicator would be $\chi^2 = 0.61$. The degree of freedom being $5-1-1=3$, we would then have $\chi^2 = 0.61 > \chi^2_\alpha = 0.58$, the hypothesis of a Poisson distribution would then not be acceptable.

Values i	0	1	2	3	≥ 4
Occurrences (O_i^x)	10	10	14	8	8
Estimated (E_i^x)	6.77	13.53	13.53	9.02	7.14

Table 13.5. *Chi-square (χ^2) test: a bad partition*

13.4.3. *Kolmogorov–Smirnov test*

The K-S test aims to check if a given sample has a certain given distribution. It is based on the fact that in case of agreement, the difference between the CDF of the target distribution (denoted by $F_L(x)$) and the empirical CDF obtained from the sample (denoted by $F_O(x)$) should be small. This leads to the following procedure:

1) order the n observations (the x_i) of the sample $\{x_1, x_2, \ldots, x_n\}$ so that $x_1 \leq x_2 \leq \cdots \leq x_n$;

2) get the empirical CDF $F_O(x)$ from the sample, which gives:

$$F_o(x) = \frac{\text{Card}\{x_i / x_i \leq x\}}{n} \qquad [13.6]$$

It is actually a step function which makes an unit jump only at points $\{x_i\}_{i=1,\ldots,n}$. $F_o(x)$ can be reformulated by:

- $F_o(x) = 0, \;\; x < x_1$;
- $F_o(x) = \frac{i}{n}, \;\; x_i \leq x < x_{i+1}$ for $i = 1 \ldots n-1$;

- $F_o(x) = 1$ for $x \geq x_n$;

3) identify the maximum difference between $F_L(x)$ and $F_o(x)$:

$$\Delta_n = \max_x |F_L(x) - F_o(x)|$$

The function $F_L(x)$ being a monotonously increasing function and $F_o(x)$ a step function, we only have to calculate the differences on the points $\{x_i\}_{i=1,\ldots,n}$. Note that it is necessary to calculate $|F_L(x_i) - F_o(x_i)|$ as well as $|F_L(x_i)) - F_o(x_{i-1})|$. Indeed, at each point x_i, $F_o(.)$ is discontinuous;

4) The quantity $\sqrt{n}\Delta_n$ is then compared to a threshold value K_α. If:

$$\sqrt{n}\Delta_n < K_\alpha$$

then there is good agreement between the distribution F_L and the sample data with a confidence level of confidence at $1 - \alpha$. If not, the hypothesis (the data have distribution F_L) is to be rejected with the same degree of confidence.

The threshold value K_α also depends on n. For a degree of confidence at 95% and for $n > 35$, we have $K_{0.05} = 1.36$.

The K-S test can also be used to test the agreement between two samples with, respectively, n_1 and n_2 observations. Let $F_1(.)$ (respectively, $F_2(.)$) be the empirical CDF obtained from the first (respectively, second) sample. The difference Δ_{n_1,n_2} is calculated as follows:

$$\Delta_{n_1,n_2} = \max_x |F_1(x) - F_2(x)|$$

The test is done by checking the following inequality:

$$\sqrt{\frac{n_1 n_2}{n_1 + n_2}}\Delta_{n_1,n_2} < K_\alpha$$

As previously, if this inequality is verified, two samples can be considered as outcomes from the same distribution, with a confidence level at $1 - \alpha$, if not, this hypothesis is rejected.

13.4.4. *Comparison between the χ^2 test and the K-S test*

The two tests have the following differences:

– the χ^2 test allows us to estimate the parameters of the distribution by using the same data, whereas the K-S test claims a distribution defined elsewhere;

– the K-S test deals with empirical CDF which is a brute translation of the sample data, whereas The χ^2 test proceeds to an artificial interpretation through their partitioning;

– for a K-S test, the entire sample should be memorized, whereas for a χ^2 test, we need simply keep K counters associated with the K intervals.

For both tests, it is interesting to note that we can resort to some specialized software, such as the `R` language (a public-domain software) to carry out these algorithmic operations.

14

Markov Process

In this chapter, we study the Markov process, which is an important class among the stochastic systems. The principal characteristic of the Markov processes is the conditional independence between the future and the past if the present is known. This conditional independence makes this kind of process sufficiently general to model a large number of concrete cases and sufficiently simple to be mathematically tractable. So the Markov processes occupy a very significant place in the theory of stochastic processes and are used as a popular tool for the design and dimensioning of computer systems and networks. For an in-depth study, the book of Cinlar [CIN 75] is a classical one. The readers can also consult [BRE 05, BOL 06]. It is interesting to quote the work [FAY 99] which is devoted to the random walks in the first quadrant, with many applications in various fields. Francophone readers can also consult [HEC 03].

14.1. Stochastic process

Modern computer systems are complex systems which evolve dynamically in time. Consider an Internet router by focusing on the number of packets in transit; it is easy to imagine that this number evolves under the combined effects of the submitted traffic and the routing capacity of the router. This number is thus random and evolves in time. This evolution has to be described by a sequence of random variables (r.v.), which are chronologically ordered.

The stochastic processes and the associated mathematical tools offer a means for the description and study of this kind of dynamic systems.

DEFINITION 14.1.– *A* stochastic process $\{X_t, t \in \mathcal{T}\}$ *is a set of r.v. which are indexed by* t *and defined on a same probability space*.

By definition, the *state space* ($\mathcal{E}$) of a process is the set of values that X_t can take, i.e. $\forall\, t \in \mathcal{T}, X_t \in \mathcal{E}$. According to the nature of $\mathcal{T}$ and that of $\mathcal{E}$, a stochastic process can be divided into four different forms:

Performance Evaluation by Simulation and Analysis with Applications to Computer Networks, First Edition. Ken Chen. © ISTE Ltd 2015. Published by ISTE Ltd and John Wiley & Sons, Inc.

– when $\mathcal{E}$ is finite or countably infinite (respectively, $\mathcal{E}$ continuous), it is called a *chain* (respectively, *process*);

– when $\mathcal{T} \subset \mathbb{N}$ (respectively, $\mathcal{T} \subset \mathbb{R}$), it is said to be *discrete time* (respectively, *continuous time*).

In the remainder of this chapter, we consider only chains (i.e. $\mathcal{E}$ is finite or countably infinite), with both discrete time and continuous time.

By representing t the time and X_t the state of a system (e.g. the number of customers), the process describes the evolution in time of the system.

EXAMPLE 14.1.– Counting process: $\{N(t), t \geq 0\}$, where $N(t) \in \mathbb{N}$ is the number of occurrences of an event (e.g. arrival of a customer) from the beginning (0) till the instant t. By definition, $N(0) = 0$. The quantity $N(t) - N(s)$, with $t > s \geq 0$, represents the number of occurrences during $]s, t]$.

For each $t \in \mathcal{T}$, X_t is a random variable. The simplest case is that where X_t is an independent and identically distributed (i.i.d.) r.v. However, the evolution of a real system is generally conditioned by its past. This means that, when we seek to model the dynamic behavior of a system with a stochastic process $\{X_t, t \in \mathcal{T}\}$, which represents the evolution in time of the system, the X_t are generally dependent. In other words, the distribution of the state at a certain instant, say t for the state X_t, may depend on all of the anterior X_u ($u < t$). These kind of studies are, of course, very difficult.

The Markov process offers a good trade-off between the timely dependency and the tractability. The principal idea consists of considering that all the *past* is summarized in the *present*. In the remainder of this chapter, we will study this class of stochastic processes exclusively.

14.2. Discrete-time Markov chains

14.2.1. *Definitions*

Consider a stochastic process $X = \{X_n; n \in \mathbb{N}\}$ in discrete time and having values in a discrete space E (either finite or countably infinite), i.e. for each n, X_n is a random variable with values in E, where E represents the state

space of the process X and X_n represents the state of the process at time n; the process is in state j at time n if $X_n = j$.

DEFINITION 14.2.– *The process* $X = \{X_n; n \in \mathbb{N}\}$ *is called a* discrete-time Markov chain *if, for all* $j \in E$ *and all* $n \in \mathbb{N}$*, we have*:

$$P[X_{n+1} = j/X_0, \ldots, X_n] = P[X_{n+1} = j/X_n] \qquad [14.1]$$

Equation [14.1] is called a Markov property. *If, moreover, the quantity* $P[Xn + 1 = j/Xn]$ *does not depend on* n*, i.e.*:

$$P[X_{n+1} = j/X_n = i] = P(i, j) \qquad (i, j) \in E^2,\ n \in \mathbb{N}, \qquad [14.2]$$

the chain is time-homogeneous *(for the sake of simplicity, we will use the short form* homogeneous *by omitting* time*). The matrix* $\boldsymbol{P}$ *defined by*:

$$\forall (i, j) \in E^2, [\boldsymbol{P}]_{ij} = P(i, j) \qquad [14.3]$$

is called the transition matrix *of the Markov chain* $\{X_n\}$.

Subsequently, we will use the short form *Markov chain* instead of *discrete-time Markov chain*, and we study only the homogeneous Markov chains unless otherwise specified. We also denote $[\boldsymbol{P}]_{ij}$ by $\boldsymbol{P}_{ij}$.

The transition matrix $\boldsymbol{P}$ of any Markov chain has the following properties:

– for all (i, j) in $E \times E$, $\boldsymbol{P}_{ij} \geq 0$;

– for all i in E, $\sum_{j \in E} \boldsymbol{P}_{ij} = 1$.

Conversely, we have the following definition:

DEFINITION 14.3.– *Let* $\boldsymbol{P}$ *be a square matrix with elements* $\boldsymbol{P}_{ij}$ *defined for all* (i, j) *in* $E \times E$. $\boldsymbol{P}$ *is a* Markov matrix *(or* transition matrix*) if*:

– for all (i, j) in $E \times E$, $\boldsymbol{P}_{ij} \geq 0$;

– for all i in E, $\sum_{j \in E} \boldsymbol{P}_{ij} = 1$.

EXAMPLE 14.2.– Here is a model for *coin tossing*. We are interested by the cumulative number of *head* after the nth trial, denoted by N_n, The state space is $\mathbb{N}$, with:

$$P(i,j) = P[N_{n+1} = j / N_n = i] = \begin{cases} p \text{ if } j = i+1 & \text{(head)} \\ q \text{ if } j = i, & \text{(tail)} \\ 0 \text{ for } j \neq i, j \neq i+1 & \text{(no such transition).} \end{cases}$$

The transition matrix is thus:

$$\boldsymbol{P} = \begin{bmatrix} q & p & & & 0 \\ & q & p & & \\ & & q & p & \\ & & & \ddots & \ddots \\ 0 & & & & \end{bmatrix}.$$

EXAMPLE 14.3.– A system with two states. It is a generic model which can be adapted to many systems. Here is the model:

– there are two states, denoted, respectively, 1 and 2, $E = \{1, 2\}$;

– the transition matrix $\boldsymbol{P}$ is given by:

$$\boldsymbol{P} = \begin{bmatrix} p_1 & q_1 \\ q_2 & p_2 \end{bmatrix}$$

with $0 \leq p_1 \leq 1, p_1 + q_1 = 1$ and $0 \leq p_2 \leq 1, p_2 + q_2 = 1$;

– the parameter p_1 (respectively, p_2) gives the probability of remaining in state 1 (respectively, 2), whereas q_1 (respectively, q_2) gives the probability of leaving this state (thus, going toward the opposite state).

Here are two examples of application of this model in telecommunications field:

1) *A transmission channel under disturbance with two levels of noise intensity*: when we want to model the noise in a transmission channel (e.g. a wireless link), the simplest model is that of a channel under uniform disturbance, which is characterized by a single parameter which is the loss rate of the channel. However, this model may be too simplistic since often

disturbance is not uniformly observed. We can then use a model with two modes, each one having its own loss rate. To complete the model, we have to give the probability of remaining, or not, in the current mode.

2) *A model for phone-call traffic*: very schematically, each sense of a telephone conversation, under normal condition, is composed of a succession of couples "speech/silence". We can thus adopt a model with two modes: active versus silence.

14.2.2. *Properties*

14.2.2.1. *Chapman–Kolmogorov equation*

Let us consider now the probability of reaching state i_m in m steps from $X_n = i_0$ and by following the path $i_0 \to i_1 \to \cdots \to i_m$. This joint probability is obtained with

$$\begin{aligned} & P[X_{n+1} = i_1, \ldots, X_{n+m} = i_m \,/\, X_n = i_0] \\ = \; & P[X_{n+1} = i_1 / X_n = i_0]\, P[X_{n+2} = i_2 / X_{n+1} = i_1] \\ & \ldots P[X_{n+m} = i_m / X_{n+m-1} = i_{m-1}] \\ = \; & P(i_0, i_1)\, P(i_1, i_2) \ldots P(i_{m-1}, i_m) \end{aligned}$$

by jointly using the conditional probability and the Markov property from X_{n+1}, and then iteratively till X_{n+m-1}.

We can apply this property in particular at the initial state X_0. Let $\nu(i) = P[X_0 = i]$ the probability the chain is in state i at instant 0, the vector $\boldsymbol{\nu} = [\nu(i)]_{i \in E}$ yields the *initial distribution* of X. We have:

$$P[X_0 = i_0, X_1 = i_1, \ldots, X_n = i_n] = \nu(i_0) P(i_0, i_1) \ldots P(i_{n-1}, i_n)$$

We can conclude from it that the evolution of Markov chain is completely determined by its initial distribution $\boldsymbol{\nu}$ and its transition matrix $\boldsymbol{P}$.

We want to calculate the probability to reach state i_m in m steps from $X_n = i_0$, i.e. $P[X_{n+m} = i_m / X_n = i_0]$. Let us start with $m = 2$:

$$\begin{aligned} & P[X_{n+2} = j / X_n = i] \\ &= \sum_{k \in E} P[X_{n+2} = j / X_{n+1} = k] P[X_{n+1} = k / X_n = i] \\ &= \sum_{k \in E} P(j,k) P(k,i) = \boldsymbol{P}^2(i,j). \end{aligned}$$

By following the same approach, we obtained:

$$P[X_{n+m} = j / X_n = i] = \boldsymbol{P}^m(i,j) \qquad [14.4]$$

Recall that $\boldsymbol{P}^m(i,j)$ is the component (i,j) of the matrix $\boldsymbol{M} = \boldsymbol{P}^m$ (i.e. m-tuple product of $\boldsymbol{P}$) and not $[\boldsymbol{P}(i,j)]^m$. As we have:

$$P[X_{n+m} = j / X_0 = i] = \sum_{k \in E} P[X_{n+m} = j / X_m = k] P[X_m = k / X_0 = i]$$

by applying the preceding relation, we obtain readily:

$$\boldsymbol{P}^{m+n}(i,j) = \sum_{k \in E} \boldsymbol{P}^m(i,k) \boldsymbol{P}^n(k,j) \qquad [14.5]$$

This equation is known as *Chapman–Kolmogorov equation*.

14.2.2.2. *Generalization*

The *memoryless* property of a Markov chain is conserved for any bounded function taking the Markov chain as parameter.

THEOREM 14.1.– Let $X = \{X_n\}_{n \in \mathbb{N}}$ be a Markov chain and $f(X_n, X_{n+1}, \dots)$ be a bounded function of $X_n, X_{n+1}, \dots$:

$$E[f(X_n, X_{n+1}, \dots) / X_0, \dots, X_n] = E[f(X_n, X_{n+1}, \dots) / X_n] \qquad [14.6]$$

and:

$$E[f(X_n, X_{n+1}, \dots) / X_n = i] = E[f(X_0, X_1, \dots) / X_0 = i] \qquad [14.7]$$

14.2.3. *Transition diagram*

A Markov chain can be graphically represented by a *directed graph* called *transition diagram*. The transition diagram of a Markov chain with transition matrix is built as follows:

– each state is represented by a node (vertex), being marked with the name of the state;

– for each $\boldsymbol{P}_{ij} > 0$, a *directed* arc (edge) goes from the node (state) i to the node (state) j, the value of $\boldsymbol{P}_{ij} > 0$ is put near this arc. In the particular case of $\boldsymbol{P}_{ii} > 0$, draw an arc which is turned back to the state i.

Figure 14.1 provides an illustration through two chains: the first chain having two states ($E = \{1, 2\}$) and the second chain having three states ($E = \{a, b, c\}$). Their respective transition matrices are given as follows:

$$\boldsymbol{D} = \begin{bmatrix} p_1 & q_1 \\ q_2 & p_2 \end{bmatrix}, \qquad \boldsymbol{T} = \begin{bmatrix} 0 & p_{ab} & p_{ac} \\ p_{ba} & 0 & 0 \\ 0 & p_{cb} & 0 \end{bmatrix}.$$

We note easily that from the transition matrix of a Markov chain, we can draw its transition diagram, and, conversely, the transition diagram allows us to determine the transition matrix. A Markov chain can thus be defined through any of the two forms. The transition diagram offers a direct visualization of the relations between the states; it facilitates thus the intuitive analysis.

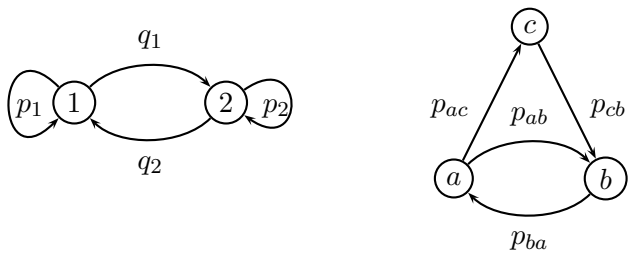

Figure 14.1. *Transition diagram of two Markov chains*

14.2.4. *Classification of states*

Let T_1 be the instant of the next visit to a state, say j. Due to the Markov property, we can consider it as the first return to j by taking j as the initial state ($X_0 = j$).

DEFINITION 14.4.– *The state j is* recurrent *if*

$$P[T_1 < \infty / X_0 = j] = 1. \qquad [14.8]$$

otherwise, it is transient.

A recurrent state is *null recurrent* if $E[T_1/X_0 = j] = \infty$; otherwise, it is *positive recurrent*.

Roughly speaking, a recurrent state is a state which will be visited an infinite number of times, which is not the case of a transient state; a positive recurrent state will be *regularly* visited, with a certain frequency, whereas for a null recurrent state, the next visit can be indefinitely far away.

DEFINITION 14.5.– *The* period *of a (recurrent) state j is the greatest common divisor (GCD), denoted δ, of all n such that $\boldsymbol{P}^n(j,j) = P[X_n = j/X_0 = j] > 0$. If $\delta \geq 2$, the state j is* periodic*; otherwise, it is* aperiodic.

We now will classify the states according to their relations with other states.

DEFINITION 14.6.– *A state j is* accessible *from another state i if $\exists\, m \in \mathbb{N}, \boldsymbol{P}^m(i,j) > 0$; we write then $i \to j$. We say i and j* communicate *if $i \to j$ and $j \to i$, and we write $i \leftrightarrow j$.*

The relation $\leftrightarrow$ is by definition *symmetric*; it is easy to check that it is also *reflexive*. This relation is also *transitive* due to the Chapman–Kolmogorov equation. Thus, $\leftrightarrow$ is an equivalence relation, which allows us to define equivalence classes on E.

DEFINITION 14.7.– *Let X be a Markov chain with state space E. Consider $F \subset E$:*

– F is *closed* if

$$\sum_{j \in F} P(i,j) = 1 \quad \forall i \in F,$$

i.e. no state outside of F could be accessible *from a state in F*;

– in particular, if F is reduced to a single state $\{j\}$, j is an *absorbing* state;

– a closed subset F is *irreducible* if any state in F can be reached from any other state in F. In other words, a closed subset is irreducible if it does not contain a smaller closed subset;

– E is by this definition closed. If, in addition, E does not contain any closed subset $F \subset E$, i.e. every state of E communicates with any other state, the Markov chain is said to be *irreducible*.

We will admit the following result.

THEOREM 14.2.– The states of an irreducible Markov chain have all the same nature: all transient, all null recurrent or all positive recurrent. If they are periodic, then they all have the same period.

14.2.5. *Stationarity*

We will examine the behavior in limiting cases of the Markov chains, in particular, the irreducible and aperiodic chain. To begin, we admit the following result.

THEOREM 14.3.– We have:

– if j is transient, null or recurrent, then for all $i \in E$:

$$\lim_{n\to\infty} \boldsymbol{P}^n(i,j) = 0$$

– if j is positive recurrent and aperiodic, then there exists a constant $\pi(j)$ such that:

$$\lim_{n\to\infty} \boldsymbol{P}^n(j,j) = \pi(j) > 0$$

and for every $i \in E$:

$$\lim_{n\to\infty} \boldsymbol{P}^n(i,j) = F(i,j)\pi(j)$$

where $F(i,j)$ is the probability that the state j is visited at least once from state i.

THEOREM 14.4.– Let X be an *irreducible* and *aperiodic* Markov chain. All states of X are *positive recurrent* if and only if the system of linear equations:

$$\begin{cases} \pi(j) & = \sum_{i \in E} \pi(i) P(i,j),\ j \in E \\ \sum_{j \in E} \pi(j) & = 1 \end{cases} \quad [14.9]$$

has one solution. If such a solution $\boldsymbol{\pi}$ exists, then it is strictly positive and it is unique, and we have:

$$\pi(j) = \lim_{n \to \infty} \boldsymbol{P}^n(i,j), \quad \forall\ (i,j) \in E^2$$

If $\boldsymbol{\pi}$ exists, the Markov chain eventually reaches its *stationary regime*, and the vector $\boldsymbol{\pi}$ gives the *stationary distribution* of the chain. Indeed, $\boldsymbol{\pi}(i)$ is the probability, after a transition period, that the chain is in state i. $\boldsymbol{\pi}$ is thus a key characteristic of a Markov chain.

EXAMPLE 14.4.– The evolution of the number of IP (Internet Protocol) packets in a router can be modeled, under certain conditions, by a Markov chain. Assume that the router can keep up to L packets. The router has to reject newly arrived packets if its buffer is full. We assume that there is also a signaling system which is triggered if the occupation of the buffer exceeds a certain threshold S ($S < L$). Two of the indicators are particularly interesting, namely:

– P_r probability (ratio) of rejection;

– P_s probability (frequency) of signaling system triggering.

The state space is clearly $\{0, \ldots, L\}$. Without actually modeling the system, we simply assume that we got the stationary distribution $(\pi(0), \ldots, \pi(L))$. Then, the two indicators P_r and P_s can be simply obtained with:

– $P_r = \pi(L)$, probability of having a full buffer;

– $P_s = 1 - \sum_{i=0}^{S-1} \pi(i)$, probability that the packet number is equal to or greater than the threshold value (S).

14.2.6. *Applications*

14.2.6.1. *Introduction*

The Markov chains constitute a very popular modeling tool. We consider here only the irreducible, aperiodic and positive recurrent Markov chains, i.e. those Markov chains that have a stationary distribution.

Before dealing with the modeling by Markov chain, we want to present very briefly the three steps that we will follow.

14.2.6.1.1. Step 1: Modeling

We have to first set up a Markov chain model for the system. It is a complex step because we usually have to accept simplifications in order to get the Markov property. If the distortions related to the real system are too strong, the Markov model would be useless.

The model is generally built for a target goal. Taking a hot-line support center with a capacity of L simultaneous calls, we can be interested in the number of rejected calls which constitutes a significant indicator of satisfaction in order to assess the performance of the center, as well as to identify the possible need for additional investment. The model is a chain comprising $L+1$ states, where each state i ($0 \leq i \leq L$) corresponds to the situation where i calls are going on. A delicate point consists of assuring the Markov property of the transition from one state to another: we have to assume that the calls follow a Poisson process and durations of calls are exponentially distributed. Another delicate point lies in the estimation of the transition probabilities.

Outcome of this step: a Markov chain characterized by its transition matrix.

14.2.6.1.2. Step 2: Get the stationary distribution

This step is *a priori* very classical because we simply have to, theoretically at least, solve a system of linear equations. We are likely to solve a system which has a large number of states for reasons of faithfulness. This issue is not covered here. However, we want to emphasize this point, which is seldom met in a school context but much often in real world.

Outcome of this step: stationary distribution of the chain.

14.2.6.1.3. Step 3: Exploitation

This last step should be performed jointly with step 1 (modeling). Indeed, it depends, as the modeling, on the goal of the study.

Let us take again the example of the hot-line call center, with its stationary distribution $\boldsymbol{\pi} = (\pi(0), \pi(1), \ldots, \pi(L))$. $\pi(L)$ gives the probability that a new call has to be discarded. $\pi(0)$ gives the probability that the hot-line center does not receive any calls. From this information, technical decisions and/or those of other natures (commercial, etc.) can be taken. For example:

– a too big $\pi(L)$ (at least bigger than the competitors) suggests a lower level of satisfaction of the customers and a need of additional resources (more operators and/or better training of them, etc.);

– an important $\pi(0)$ suggests a possible decrease of L, thus less resources (typically less operators devoted to the hot-line support).

In both cases, we have to deal with *new* models, at least with parameter changes, and it is necessary to take again the process starting from step 1 (or step 2 if only parameters are changed). We want to highlight this interaction between modeling and its exploitation.

Results of this step:

– diagnosis on the system through various information provided by the model;

– possible modification of the system (architecture and/or configuration).

14.2.6.2. *Modeling*

We present here a simple example, which is a generic model for a system which evolves among three modes. One can use it, for instance, to model the traffic received by an Internet router: low, average or strong.

The state space, E, of this system is obvious: we denote, for the sake of simplicity, the three states by $0, 1, 2$, i.e. $E = \{0, 1, 2\}$.

To be able to model it by a Markov chain, we must adopt the assumption that the transition from one state to another depends only on those two states (present and future) and that the past plays no role there. This could be a

possible distortion compared to the reality, since it is not always verified, in an exact way, in practice. The Markov assumption is acceptable for the systems where the influence of the past is weak. If the weight of the past becomes indeed significant, the Markov model is no longer suitable. We must also assume that the transition probability is invariant in time in order to make the chain homogeneous. Once again, it is necessary to assess the faithfulness of this assumption to the reality.

Under these assumptions, we deal with a Markov chain having three states, $X = \{X_n\}_{n>0}, X_n \in E$. To characterize this chain completely, it is necessary to determine its transition matrix. This is obtained in practice either by statistical measurements (the sequence $\{X_n\}_{n>0}$ can be obtained through the analysis of traffic trace) or by an estimated parameter setting. For example, p_{01} can be estimated from the percentage of the jumps from state 0 to state 1. Alternatively, this same parameter can also correspond to a particular scenario imagined by the designer. Let us now suppose that this matrix, $\boldsymbol{P}$, is given by:

$$\boldsymbol{P} = \begin{pmatrix} 0.7 & 0.2 & 0.1 \\ 0.4 & 0.4 & 0.2 \\ 0.6 & 0.3 & 0.1 \end{pmatrix}$$

The transition diagram (see Figure 14.2) allows us to quickly, and intuitively, check that the chain is irreducible because all the states communicate with each other. We can easily check that it is aperiodic.

14.2.6.3. *Steady-state solution*

The stationary distribution of the chain, denoted $\boldsymbol{\pi}$, is obtained by solving the following system of linear equations:

$$\begin{cases} \boldsymbol{\pi}\boldsymbol{P} = \boldsymbol{\pi} \\ \sum_{j \in E} \pi_j = 1 \end{cases} \qquad [14.10]$$

where π_j is the jth component of $\boldsymbol{\pi}$. This gives:

$$\begin{cases} \pi_0 = 0.7\pi_0 + 0.4\pi_1 + 0.6\pi_2 \\ \pi_1 = 0.2\pi_0 + 0.4\pi_1 + 0.3\pi_2 \\ \pi_2 = 0.1\pi_0 + 0.2\pi_1 + 0.1\pi_2 \end{cases}$$

under the condition: $\pi_0 + \pi_1 + \pi_2 = 1$. The solution is $\boldsymbol{\pi} = (26/61, 25/61, 10/61)$.

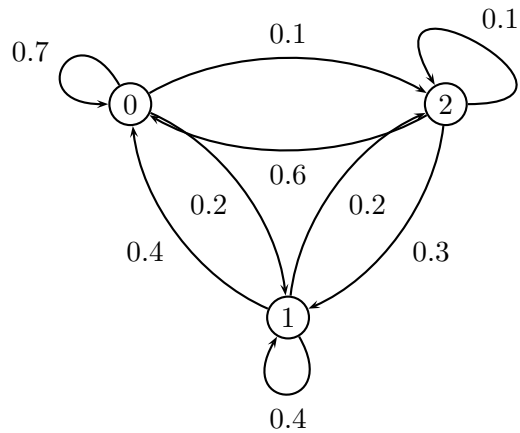

Figure 14.2. *Transition diagram of a Markov chain with three states*

14.2.6.4. *Exploitation*

We can conclude for instance that, on average, the state 0 represents about 43%, or, in an equivalent way, a new user finds the system in state 0 with a probability of 43%. If the state 0 represents the low traffic state of a router, it is a router which is weakly loaded.

14.3. Continuous-time Markov chain

14.3.1. *Definitions*

Consider a stochastic process $X = \{X_t; t \in \mathbb{R}_+\}$, $X_t \in E \; \forall t \in \mathbb{R}_+$, with continuous time ($t$) and discrete state space (E being finite or countably infinite). $X_t = j$ means the process is in state j at time t.

DEFINITION 14.8.– *The process* $X = \{X_t; t \in \mathbb{R}_+\}$ *is a* continuous-time Markov chain (CTMC) *if* $\forall j \in E$ *and* $\forall (s,t) \in \mathbb{R}_+ \times \mathbb{R}_+$:

$$P[X_{t+s} = j / X_u; u \leq t] = P[X_{t+s} = j / X_t]. \qquad [14.11]$$

Equation [14.11] is called a *Markov property*. The future and the past are thus conditionally independent, knowing its present X_t. For the sake of simplicity, in the remainder of this chapter, we will use the term *Markov process* instead of *continuous-time Markov chain* or *CTMC*.

In general, $P[X_{t+s} = j/X_t]$ depends not only on the duration s which separates the present (X_t) and the future (X_{t+s}), but also on the instant t at which the present is located. If this quantity does not depend on t, then the process is known as *time homogeneous*. We will use the term *homogeneous* by omitting the word *time*. In the remainder of this chapter, we will study only the homogeneous Markov processes, unless otherwise specified. We can then define, for all i, j in E and all $s \geq 0$, the quantity:

$$P[X_{t+s} = j/X_t = i] = P_s(i, j) \qquad [14.12]$$

which is independent of t. For a given couple (i, j), the function $s \to P_s(i, j)$ is called *transition function*.

The set of the transition functions, $P_s(i, j), (i, j) \in E^2$, constitutes a matrix defined by

$$\forall (i, j) \in E^2,\ [\boldsymbol{P_s}]_{ij} = P_s(i, j).$$

EXAMPLE 14.5.– Poisson process (see Chapter 8). This process X_t with values in $\mathbb{N}$ ($E = \mathbb{N}$) is defined by:

$$P_s(i, j) = P[X_{t+s} = j/X_t = i] = \begin{cases} e^{-\lambda s} \times \frac{(\lambda s)^{(j-i)}}{(j-i)!} & \text{if } j \geq i, \\ 0 & \text{if } j < i \end{cases}$$

where λ is the rate of the process, with the initial condition $P[X_0 = 0] = 1$. The transition function is as follows:

$$\boldsymbol{P}_t = \begin{bmatrix} P_t(0) & P_t(1) & P_t(2) & P_t(3) & \dots \\ & P_t(0) & P_t(1) & P_t(2) & \dots \\ & & P_t(0) & P_t(1) & \dots \\ & & & . & \dots \\ 0 & & & & \dots \end{bmatrix}$$

The Poisson process occupies a very significant place in telecommunications because it allows us to get accurate modeling of many

types of incoming traffics: phone calls, number of unique visits to a Website, number of mails received by a mail server, etc.

14.3.2. *Properties*

14.3.2.1. *Elementary properties*

We notice that the matrix $\boldsymbol{P}_t = [P_t(i,j)]$ has the following properties for all i, j in E and all t, s in $\mathbb{R}_+$:

1) $P_t(i,j) \geq 0$;

2) $\sum_{k \in E} P_t(i,k) = 1$;

3) $\sum_{k \in E} P_t(i,k) P_s(k,j) = P_{t+s}(i,j)$, i.e. $\boldsymbol{P}_{t+s} = \boldsymbol{P}_t \boldsymbol{P}_s$.

These properties are similar to their equivalent in the discrete case (Markov chains); the first two properties come from the fact that $\boldsymbol{P}_t$ is a Markov matrix and the third property is the Chapman–Kolmogorov equation for the continuous-time case.

14.3.2.2. *Other properties*

By introducing $\boldsymbol{P}_0 = \boldsymbol{I}$, where $\boldsymbol{I}$ is the identity matrix, the family of these matrices $\{\boldsymbol{P}_t;\ t \geq 0\}$ forms a semigroup called *transition semigroup*.

Just like for the Markov chains, it is possible to show that the Markov property implies also some more powerful properties. By denoting $\boldsymbol{\nu}$ the initial distribution of X, i.e. $\nu(i) = P[X_0 = i]$, the Markov property implies the following:

THEOREM 14.5.– For all $n \in \mathbb{N}$ and all sequence of positive real: $0 = t_0 < t_1 < \cdots < t_n$, as well as all sequence of the elements of E: $i_0, i_1, \ldots, i_n$:

$$\begin{aligned} P[X_0] &= P[i_0, X_{t_1} = i_1, \ldots, X_{t_n} = i_n] \\ &= \nu(i_0) P_{t_1 - t_0}(i_0, i_1) \ldots P_{t_n - t_{n-1}}(i_{n-1}, i_n), \end{aligned}$$

i.e. the behavior of a Markov process is characterized by its initial distribution $\boldsymbol{\nu}$ and its transition semigroup $\{\boldsymbol{P}_t\}_{t \geq 0}$.

14.3.3. *Structure of a Markov process*

Let ω be an element (an outcome) of the probability space Ω, $\{X_t(\omega)\}_{t\in\mathbb{R}_+}$ describes a trajectory of X. In the following, we will follow such a trajectory by omitting ω.

Without going into the mathematical details, we impose the following continuity condition concerning the transition functions:

$$\lim_{t\downarrow 0} P_t(i,j) = \delta_{i,j} \quad (i,j) \in E^2, \qquad [14.13]$$

where $\delta_{i,j}$ is the notation of Kronecker: $\delta_{i,i} = 1$ and $\delta_{i,j} = 0,\ i \neq j$. Then, P_t is called *a standard transition function*. Subsequently, P_t will be such a function. We admit the following result:

THEOREM 14.6.– For all i, j in E, the function $t \rightarrow P_t(i,j)$ is 0 either everywhere or nowhere in $(0,\infty)$. In both cases, it is continuous everywhere.

Let t be a fixed instant, and define $W_t = \inf\{s > 0 / X_{t+s} \neq X_t\}$, which is the duration of the interval where the process continues to remain in state X_t after the instant t. It can be shown that W_t has an *exponential distribution*.

THEOREM 14.7.– For all $i \in E$ and all $t \geq 0$, $u \geq 0$, there is a $\lambda(i) \in [0,\infty]$ such that:

$$P[W_t > u / X_t = i] = e^{-\lambda(i)u} \qquad [14.14]$$

where $e^{-\lambda(i)u} = 0$ for all $u \geq 0$ if $\lambda(i) = \infty$.

This result shows the *memoryless* behavior of the Markov processes. Indeed, if, at a given moment, a process X is in state i, then the time that it continues to remain in this state is independent of the time that it spent in this same state, as if it had lost memory about its past. According to the value of the associated $\lambda(i)$, a state can be classified as follows.

DEFINITION 14.9.– A state i is:

– *stable* if $0 < \lambda(i) < \infty$;

– *absorbing* if $\lambda(i) = 0$;

– *instantaneous* if $\lambda(i) = \infty$.

It is easy to see that:

– if i is *stable*, then $P[0 < W_t < \infty / X_t = i] = 1$, i.e. if the process X enters in state i, it will remain there for a finite and non-null duration;

– if i is *absorbing*, then for all $u \geq 0$, $P[W_t > u / X_t = i] = 1$, i.e. if the process X enters in state i, it will never again leave it;

– if i is *instantaneous*, then $P[W_t = 0 / X_t = i] = 1$, i.e. the process X leaves state i as soon as it enters there.

The instantaneous states are somewhat troublesome because the process will have an indefinite number of jumps once entered an instantaneous state. It was shown that the process X does not have an instantaneous state in the following cases:

– E is finite;

– the function $t \to X_t(\omega)$ is *right continuous* for *almost all* $\omega \in \Omega$.

In the following, we suppose that the process X does not have instantaneous state (see Figure 14.3).

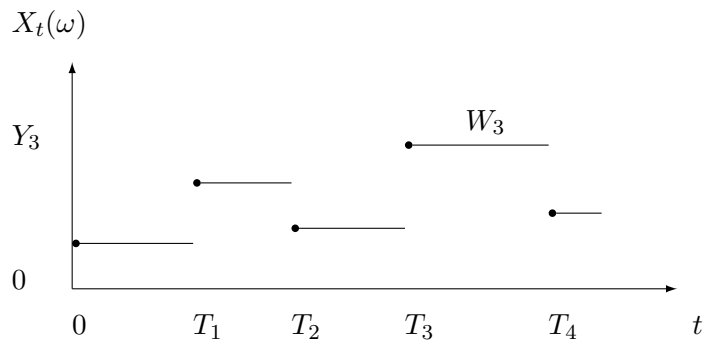

Figure 14.3. *A sequence of Markov process X with right continuity*

We are now interested more particularly in the jumping instants. Let $T_0 = 0$ and define $\{T_n\}_{n\in\mathbb{N},n>0}$ as follows:

$$T_{n+1} = \inf\{t > T_n; X_t \neq X_{T_n}\} \quad [14.15]$$

where T_n are actually the jumping instants, i.e. the instants at which the process passes from one state to another (in fact, they are the so-called *stopping times* because their definition is based on the past only). Let us note:

$$\begin{cases} Y_n &= X_{T_n} \\ W_n &= T_{n+1} - T_n. \end{cases} \quad [14.16]$$

where Y_n is the state of X after the nth jump, and X keeps in the state Y_n during the period $]T_n, T_{n+1}]$ whose duration will be denoted by W_n. The following result shows that:

– $\{Y_n, n \in \mathbb{N}\}$ forms a Markov chain;

– W_n has an exponential distribution whose parameter depends on the state Y_n.

THEOREM 14.8.– For all $n \in \mathbb{N}$, $j \in E$ and $u \in \mathbb{R}_+$, we have:

$$P[Y_{n+1} = j, W_n > u/Y_0, \ldots, Y_n; T_0, \ldots T_n] = Q(i,j)e^{-\lambda(i)u} \quad [14.17]$$

if $\{Y_n = i\}$ takes place. Moreover, $\{Y_n, n \in \mathbb{N}\}$ forms a Markov chain with transition matrix $\boldsymbol{Q} = [Q(i,j)]$.

The chain $Y = \{Y_n, n \in \mathbb{N}\}$ is called the *embedded chain* of the process X_t. We will see in the following section how to obtain explicitly the vector $\boldsymbol{\lambda}$ and the matrix $\boldsymbol{Q}$.

It comes from the previous theorem that, as the transition goes necessary to one of the states $j \neq i$,

$$P[W_n > u/Y_n = i, Y_{n+1} = j] = e^{-\lambda(i)u}, \quad [14.18]$$

in other words, the sojourn time in a state depends only on the current state and not on the following state.

14.3.4. *Generators*

Now, we will focus solely on *regular* Markov process.

DEFINITION 14.10.– *A Markov process is* regular *if, for almost all $\omega \in \Omega$, we have*:

1) $t \to X_t(\omega)$ *is right continuous*;

2) $T_\infty = \infty$.

We will formulate a relation between the transition functions $\boldsymbol{P}_t$ and the parameters $\lambda(i)$ and $Q(i,j)$.

THEOREM 14.9.– For all i, j in E, the function $t \to P_t(i,j)$ is differentiable and its derivative is continuous. The derivative at $t = 0$ is given by:

$$\left.\frac{dP_t(i,j)}{dt}\right|_{t=0} = A(i,j) = \begin{cases} \lambda(i)Q(i,j) & i \neq j, \\ -\lambda(i) & i = j. \end{cases} \quad [14.19]$$

For a given $t \geq 0$, we have:

$$\frac{dP_t(i,j)}{dt} = \sum_{k \in E} A(i,k)\, P_t(k,j) \quad [14.20]$$

and also:

$$\frac{dP_t(i,j)}{dt} = \sum_{k \in E} P_t(i,k)\, A(k,j) \quad [14.21]$$

This result is very useful. Indeed:

– if the transition function P_t is known, then through its derivative we obtain $\boldsymbol{Q}$ and $\boldsymbol{\lambda}$. Indeed, $\lambda(i) = A(i,i) = \frac{dP_0(i,i)}{dt}$. If $\lambda(i) = 0$, then $Q(i,j)$ are null for all $j \neq i$; otherwise, $Q(i,j) = \frac{A(i,j)}{\lambda(i)}$;

– conversely, if $Q(i,j)$ and $\lambda(i)$ are known, then we can deduce the matrix $\boldsymbol{A} = [A(i,j)]$. We can calculate $\boldsymbol{Pt}$ by solving:

- either the system of differential equations [14.20] called *backward Kolmogorov equation* and which can be expressed as:

$$\frac{d\boldsymbol{P}_t}{dt} = \boldsymbol{A}\boldsymbol{P}_t \qquad [14.22]$$

- or the system of differential equations [14.21] called *forward Kolmogorov equation* and which can be expressed as:

$$\frac{d\boldsymbol{P}_t}{dt} = \boldsymbol{P}_t\boldsymbol{A} \qquad [14.23]$$

The solution will be $\boldsymbol{P}_t = e^{\boldsymbol{A}t}$, where:

$$\boldsymbol{P}_t = e^{\boldsymbol{A}t} = \boldsymbol{I} + t\boldsymbol{A} + \frac{t^2}{2!}\boldsymbol{A}^2 + \cdots = \sum_{n=0}^{\infty} \frac{t^n}{n!}\boldsymbol{A}^n. \qquad [14.24]$$

DEFINITION 14.11.– *The matrix* $\boldsymbol{A} = [A(i,j)]_{(i,j)\in E^2}$ *is called the* infinitesimal generator of X.

14.3.5. *Stationarity*

The nature of a Markov process is intrinsically related to that of the embedded Markov chain.

DEFINITION 14.12.– *A Markov process is* irreducible *(respectively,* transient*,* recurrent*) if the embedded chain is* irreducible *(respectively,* transient*,* recurrent*).*

In the remaining, we will consider solely the irreducible and recurrent Markov processes. We recall that the processes in consideration are *regular*.

THEOREM 14.10.– Let X be an *irreducible* and *recurrent* Markov process. The system of linear equations:

$$\boldsymbol{\mu}\boldsymbol{A} = 0 \qquad [14.25]$$

has a single solution $\boldsymbol{\mu}$ (to a multiplying factor) which is strictly positive, i.e. $0 < \mu(i) < \infty$ for all $i \in E$, $\mu(i)$ has the form:

$$\mu(i)\lambda(i) = \nu(i) \quad i \in E \qquad [14.26]$$

where $\boldsymbol{\nu}$ is a solution of the equation $\boldsymbol{\nu} = \boldsymbol{\nu}\boldsymbol{Q}$.

By multiplying both sides of formula [14.24] by $\boldsymbol{\mu}$, such that $\boldsymbol{\mu A} = 0$, we obtain $\boldsymbol{\mu P}_t = \boldsymbol{\mu}$, i.e. $\boldsymbol{\mu}$ is an *invariant measure*. We are interested more particularly in summable measures, since it allows us to obtain an invariant probability measure.

DEFINITION 14.13.– *A recurrent Markov process is* ergodic *if it has a summable and non-null invariant measure*.

As the processes are regular, we are interested in the behavior of X far after its initial phase.

DEFINITION 14.14.– *We say that a* stationary regime *exists for a process X if*:

– the limits $\pi(j) = \lim_{t\to\infty} P_t(i,j)$, $j \in E$, exist and are independent of the initial state;

– in addition, these limits constitute a probability measure, i.e.:

$$\sum_{j\in E} \pi(j) = 1$$

We admit the following result:

THEOREM 14.11.– Let X be an *irreducible* and *recurrent* Markov process, then, for all $i \in E$, the limit:

$$\pi(i) = \lim_{t\to\infty} P[X_t = i] \qquad [14.27]$$

exists and is independent of the initial distribution. Moreover, either all the $\pi(i)$ are null, i.e. the process is said to be *null recurrent*; or all the $\pi(i)$ are strictly positive, i.e. the process is said to be *positive recurrent*.

Let us take two instants s and t; the Chapman–Kolmogorov equation gives $\boldsymbol{P}_{s+t} = \boldsymbol{P}_s \boldsymbol{P}_t$. By making $s \to \infty$, we get:

$$\boldsymbol{\pi P}_t = \boldsymbol{\pi}$$

which is valid for all $t \geq 0$. We notice that $\boldsymbol{\pi}$ is an invariant measure. By making differentiation of $\boldsymbol{\pi P}_t = \boldsymbol{\pi}$ with respect to $t = 0$, we obtain:

$$\boldsymbol{\pi A} = 0 \qquad [14.28]$$

This equation has a great utility as shown by the following result that we will admit.

THEOREM 14.12.– An irreducible and regular Markov process is ergodic if and only if the system of the following equations has a single solution $\boldsymbol{\pi}$:

$$\boldsymbol{\pi}\boldsymbol{A} = 0 \quad \text{and} \quad \sum_{j \in E} \pi(j) = 1. \tag{14.29}$$

Then, $\boldsymbol{\pi}$ is an invariant probability measure, $\boldsymbol{\pi}\boldsymbol{P}_t = \boldsymbol{\pi}$. Moreover, for all i, j in E:

$$\lim_{t \to \infty} P_t(i, j) = \pi(j)$$

We can see that a stationary regime exists for an ergodic process. The invariant measure $\boldsymbol{\pi}$ is obtained by solving the system of equations:

$$\begin{cases} \boldsymbol{\pi}\,\boldsymbol{A} = 0 \\ \boldsymbol{\pi}\,\mathbf{1} \ = 1 \end{cases} \tag{14.30}$$

where $\mathbf{1}$ is a uni-colon vector with 1 everywhere, $\boldsymbol{\pi}$ is the *stationary distribution of the process*, its ith component $\pi(i)$ can be interpreted as the fraction of time that the process remains in state i, and the equation $\boldsymbol{\pi}\boldsymbol{A} = 0$ is the steady-state equation. Indeed, for a given i, we have:

$$A(i, i)\pi(i) + \sum_{j \in E, j \neq i} \pi(j) A(j, i) = 0.$$

By noticing that (see equation [14.19]) $A(i, i) = -\sum_{j \in E, j \neq i} A(i, j)$, we have:

$$\sum_{j \in E, j \neq i} \pi(j) A(j, i) = \left[\sum_{j \in E, j \neq i} A(i, j) \right] \pi(i) \tag{14.31}$$

This equation can be interpreted as a *balance equation*. The left part of the equation gives the sum of the average numbers of transitions from state j to state i with $A(j, i)$ jumps per unit of time, and the right part gives that of leaving state i. The matrix $\boldsymbol{A}$ thus gathers all of the transition rates. For this reason, $\boldsymbol{A}$ is also called *matrix of the instantaneous transition rates*.

14.3.6. *Transition diagram*

14.3.6.1. *Constitution*

Just like for the Markov chains, we can graphically represent a Markov process by using a directed graph, also called *transition diagram*. Such a diagram is built in the following way by using the infinitesimal generator $\boldsymbol{A}$ of the related process:

– each state is represented by a node (vertex) which is marked by the name of this state;

– for each positive element $\boldsymbol{A}_{ij} > 0$ with $i \neq j$, an arc (edge) goes from the node (state) i to the node (state) j. The value of $\boldsymbol{A}_{ij} > 0$ is put near this arc.

Since for each i, we have $\boldsymbol{A}_{ii} = -\sum_j \boldsymbol{A}_{ij}$, the transition diagram thus provides all the states and all the elements of $\boldsymbol{A}$. There is an equivalence between the generator $\boldsymbol{A}$ and the transition diagram.

EXAMPLE 14.6.– Here is the infinitesimal generator $\boldsymbol{A}$ of a Markov process:

$$\boldsymbol{A} = \begin{pmatrix} -3 & 0 & 3 \\ 1 & -4 & 3 \\ 0 & 2 & -2 \end{pmatrix}$$

Its transition diagram is given in Figure 14.4.

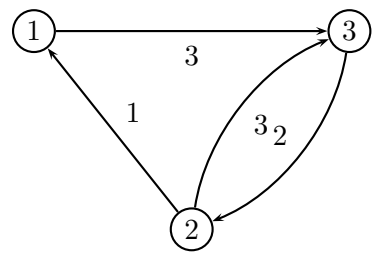

Figure 14.4. *Transition diagram of a Markov process with three states*

Compared to the setup of the transition diagrams for Markov chains (see section 14.2.3), we can observe two notable differences:

– it is made from the infinitesimal generator (the matrix $\boldsymbol{A}$), whose components $\boldsymbol{A}_{ij}$ are transition (jump) rates and not probabilities, their values can thus *exceed* 1;

– there is no return on its own state.

14.3.6.2. *Transition diagram and the balance equations*

The transition diagram offers a visual means which facilitates the setup of the balance equations. Indeed, if we take again the line i of the system of equations $\boldsymbol{\pi A} = 0$ under its form given in equation [14.31] that we reproduce as follows:

$$\sum_{j \in E, j \neq i} \pi(j) A(j,i) = \left[\sum_{j \in E, j \neq i} A(i,j) \right] \pi(i)$$

we can notice that this line can be written very easily by using the transition diagram in the following way:

– we choose a state, say i;

– we count all the arcs leaving i and we make the sum of $A(i,j)$ associated to these arcs. By multiplying this sum by $\pi(i)$, the probability of being in the state i, we obtain the term of the right-hand side: $[\sum_{j \in E, j \neq i} A(i,j)]\pi(i)$;

– we count all the arcs entering i. For each input arc, we multiply $A(j,i)$ by $\pi(j)$, the probability of having been at state j from where the transition takes place. By making the sum of these values, we obtain the term on the left-hand side: $\sum_{j \in E, j \neq i} \pi(j) A(j,i)$.

EXAMPLE 14.7.– By taking the transition diagram of the previous example (see Figure 14.4), we have:

$$\begin{cases} \pi(2) & = & 3\pi(1) \\ 2\pi(3) & = & (1+3)\pi(2) \\ 3\pi(1) + 3\pi(2) & = & 2\pi(3) \end{cases}$$

with, of course, $\pi(1) + \pi(2) + \pi(3) = 1$. We get $\boldsymbol{\pi} = (1/10, 3/10, 6/10)$.

14.3.7. *Applications*

The Markov process is a fundamental tool for modeling. It constitutes in particular the basis of the Markov queueing systems. As Markov processes take into account the time component (sojourn time in a state), they facilitate a richer and more realistic modeling, compared to discrete-time Markov chains.

We give a very simple example below. It is about a "queue" which is capable of accommodating two customers, one of which is being served and the other one of which is on standby mode. We suppose that the customers arrive with a rate of $\lambda = 2$ customers per hour. We suppose also that the average service duration is 15 min, i.e. the service (departure) rate is $\mu = 4$ customers per hour. We also suppose that the customers arrive and served individually (they thus leave the queue individually).

The state space is $E = \{0, 1, 2\}$, which corresponds to the number of customers in the queue. The transition diagram is given in Figure 14.5. The assumptions about the individual arrivals and departures prohibit the transitions other than between the neighbor states.

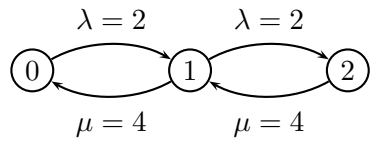

Figure 14.5. *Transition diagram of a Markov process model for a two-place queue*

The steady-state equations result directly from this diagram:

$$\begin{cases} 4\pi(1) = 2\pi(0) \\ 2\pi(0) + 4\pi(2) = 6\pi(1) \\ 2\pi(1) = 4\pi(2) \end{cases}$$

with $\pi(0) + \pi(1) + \pi(2) = 1$. This gives $\boldsymbol{\pi} = (4/7, 2/7, 1/7)$. We deduce that the rejection rate is $P_r = \pi(2) = 1/7 = 14.3\%$. If this rate is considered too high, it will be necessary to increase μ (effectiveness of processing) in order to decrease P_r, under the assumption that the arrival intensity (λ) remains unchanged.

Bibliography

[ALL 78] ALLEN A.O., *Probability, Statistics, and Queueing Theory with Computer Science Applications*, Academic Press, 1978.

[ASM 07] ASMUSSEN S., GLYNN P.W., (eds.), *Stochastic Simulation: Algorithms and Analysis*, Algorithms and Analysis, vol. 57, Springer, 2007.

[BAN 05] BANKS J., CARSON II J.S., NELSON B.L., *et al.*, *Discrete-Event System Simulation* , 4th ed., Pearson Education International, 2005.

[BEC 06] BECKER M., BEYLOT A.-L., (eds.), *Simulation des réseaux*, Hermès–Lavoisier, Paris, 2006.

[BHA 08] BHAT U.N., *An Introduction to Queueing Theory: Modeling and Analysis in Applications*, Springer, 2008.

[BOL 06] BOLCH G., GREINER S., DE MEER H., *et al.*, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*, John Wiley & Sons, 2006.

[BRE 05] BREUER L., BAUM D., *An Introduction to Queueing Theory and Matrix-Analytic Methods*, Springer, 2005.

[CHO 99] CHOI H.-K., LIMB J.O., "A behavioral model of web traffic", *Proceedings of the 7th International Conference on Network Protocols*, *1999 (ICNP'99)* pp. 327–334, 1999.

[CIN 75] CINLAR E., *Introduction to Stochastic Processes*, Prentice-Hall, Englewood Cliffs, NJ, 1975.

[COH 69] COHEN J.W., *The Single Server Queue*, North-Holland, 1969.

[COO 81] COOPER R.B., *Introduction to Queueing Theory*, 2nd ed., North-Holland, 1981.

[DAI 05] DAIGLE J.N., *Queueing Theory with Applications to Packet Telecommunication*, Springer, 2005.

[DAT 08] DATTATREYA G.R., *Performance Analysis of Queueing and Computer Networks*, CRC Press, 2008.

Performance Evaluation by Simulation and Analysis with Applications to Computer Networks, First Edition. Ken Chen. © ISTE Ltd 2015. Published by ISTE Ltd and John Wiley & Sons, Inc.

[DEC 12] DECREUSEFOND L., MOYAL P., *Stochastic Modeling and Analysis of Telecoms Networks*, ISTE, London and John Wiley & Sons, New York, 2012.

[DEG 81] DEGROOT M.H., SCHERVISH M.J., *Probability and Statistics*, 4th ed., Addison Wesley, 1981.

[FAY 99] FAYOLLE G., IASNOGORODSKI R., MALYSHEV V.A., *Random Walks in the Quarter-Plane: Algebraic Methods, Boundary Value Problems and Applications*, Springer-Verlag, 1999.

[FEL 68] FELLER W., *An Introduction to Probability Theory and its Applications*, John Wiley & Sons, New York, 1968.

[FIS 73] FISHMAN G.S., *Concepts and Methods in Discrete Event Digital Simulation*, John Wiley & Sons, 1973.

[FIS 01] FISHMAN G.S., *Discrete-Event Simulation: Modeling, Programming, and Analysis*, Springer-Verlag, 2001.

[FLO 01] FLOYD S., PAXSON V., "Difficulties in simulating the Internet", *IEEE/ACM Transactions on Networking (TON)*, vol. 9, no. 4, pp. 392–403, IEEE Press, 2001.

[GEL 98] GELENBE E., PUJOLLE G., *Introduction to Queueing Networks*, 2nd ed., John Wiley & Sons, 1998.

[HEC 03] HÊCHE J.-F., LIEBLING T.M., DE WERRA D., *Recherche opérationnelle pour ingénieurs*, PPUR, vol. 2, 2003.

[INC 07] INCE N., BRAGG A., (eds.), *Recent Advances in Modeling and Simulation Tools for Communication Networks and Services*, Springer, 2007.

[JAI 91] JAIN R., *The Art of Computer Systems Performance Analysis*, John Wiley & Sons, vol. 182, 1991.

[KLE 75] KLEINROCK L., *Queueing Systems, Volume 1, Theory*, John Wiley & Sons, 1975.

[KNU 81] KNUTH D.E., *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, Addison-Wesley Reading, MA, 1981.

[LEJ 11] LEJEUNE M., *Statistique: La théorie et ses applications*, Springer-Verlag, 2011.

[LeL 78] LE LANN G., "Algorithms for distributed data-sharing systems which use tickets", *Third Berkeley Workshop on Distributed Data Base and Computer Networks,* Berkeley, CA, pp. 259–272, 1978.

[MIT 04] MITZENMACHER M., "A brief history of generative models for power law and lognormal distributions", *Internet Mathematics*, vol. 1, no. 2, pp. 226–251, Taylor & Francis, 2004.

[MOO 09] MOORE D.S., MCCABE G.P., CRAIG B.A., *Introduction to the Practice of Statistics*, 6th ed., Freeman, 2009.

[PAR 88] PARK S.K., MILLER K.W., "Random number generators: good ones are hard to find", *Communications of the ACM*, vol. 31, no. 10, pp. 1192–1201, ACM, 1988.

[ROB 03] ROBERT P., *Stochastic Networks and Queues*, Springer-Verlag, 2003.

[TAK 91] TAKAGI H., *Queueing Analysis Volume 1 Vacation and Priority Systems*, Elsevier Science, 1991.

[VAN 06] VAN MIEGHEM P., *Performance Analysis of Communications Networks and Systems*, Cambridge University Press, 2006.

[WAS 04] WASSERMAN L., *All of Statistics: A Concise Course in Statistical Inference*, Springer, 2004.

[WEH 10] WEHRLE K., GÜNES M., GROSS J., (eds.), *Modeling and Tools for Network Simulation*, Springer, 2010.

[YUE 09] YUE W., TAKAHASHI Y., TAKAGI H., (eds.), "Advances in queueing theory and network applications*", Proceedings of the Biennial Seminar of the Canadian Mathematical Congress,* Calgary, Alberta, 12–27 June, 1978, Springer, 2009.

Index

Performance Evaluation by Simulation and Analysis with Applications to Computer Networks, First Edition. Ken Chen. © ISTE Ltd 2015. Published by ISTE Ltd and John Wiley & Sons, Inc.

Other titles from

in

Computer Engineering

2014

BOULANGER Jean-Louis
Formal Methods Applied to Industrial Complex Systems

BOULANGER Jean-Louis
Formal Methods Applied to Complex Systems: Implementation of the B Method

GARDI Frédéric, BENOIST Thierry, DARLAY Julien, ESTELLON Bertrand, MEGEL Romain
Mathematical Programming Solver based on Local Search

KRICHEN Saoussen, CHAOUACHI Jouhaina
Graph-related Optimization and Decision Support Systems

LARRIEU Nicolas, VARET Antoine
Rapid Prototyping of Software for Avionics Systems: Model-oriented Approaches forComplex Systems Certification

OUSSALAH Mourad Chabane
Software Architecture 1

OUSSALAH Mourad Chabane
Software Architecture 2

QUESNEL Flavien
Scheduling of Large-scale Virtualized Infrastructures: Toward Cooperative Management

RIGO Michel
Formal Languages, Automata and Numeration Systems 1: Introduction to Combinatorics on Words
Formal Languages, Automata and Numeration Systems 2: Applications to Recognizability and Decidability

SAINT-DIZIER Patrick
Musical Rhetoric: Foundations and Annotation Schemes

TOUATI Sid, DE DINECHIN Benoit
Advanced Backend Optimization

2013

ANDRÉ Etienne, SOULAT Romain
The Inverse Method: Parametric Verification of Real-time Embedded Systems

BOULANGER Jean-Louis
Safety Management for Software-based Equipment

DELAHAYE Daniel, PUECHMOREL Stéphane
Modeling and Optimization of Air Traffic

FRANCOPOULO Gil
LMF — Lexical Markup Framework

GHÉDIRA Khaled
Constraint Satisfaction Problems

ROCHANGE Christine, UHRIG Sascha, SAINRAT Pascal
Time-Predictable Architectures

WAHBI Mohamed
Algorithms and Ordering Heuristics for Distributed Constraint Satisfaction Problems

ZELM Martin *et al.*
Enterprise Interoperability

2012

ARBOLEDA Hugo, ROYER Jean-Claude
Model-Driven and Software Product Line Engineering

BLANCHET Gérard, DUPOUY Bertrand
Computer Architecture

BOULANGER Jean-Louis
Industrial Use of Formal Methods: Formal Verification

BOULANGER Jean-Louis
Formal Method: Industrial Use from Model to the Code

CALVARY Gaëlle, DELOT Thierry, SEDES Florence, TIGLI Jean-Yves
Computer Science and Ambient Intelligence

MAHOUT Vincent
Assembly Language Programming: ARM Cortex-M3 2.0: Organization, Innovation and Territory

MARLET Renaud
Program Specialization

SOTO Maria, SEVAUX Marc, ROSSI André, LAURENT Johann
Memory Allocation Problems in Embedded Systems: Optimization Methods

2011

BICHOT Charles-Edmond, SIARRY Patrick
Graph Partitioning

BOULANGER Jean-Louis
Static Analysis of Software: The Abstract Interpretation

CAFERRA Ricardo
Logic for Computer Science and Artificial Intelligence

HOMES Bernard
Fundamentals of Software Testing

KORDON Fabrice, HADDAD Serge, PAUTET Laurent, PETRUCCI Laure
Distributed Systems: Design and Algorithms

KORDON Fabrice, HADDAD Serge, PAUTET Laurent, PETRUCCI Laure
Models and Analysis in Distributed Systems

LORCA Xavier
Tree-based Graph Partitioning Constraint

TRUCHET Charlotte, ASSAYAG Gerard
Constraint Programming in Music

VICAT-BLANC PRIMET Pascale *et al.*
Computing Networks: From Cluster to Cloud Computing

2010

AUDIBERT Pierre
Mathematics for Informatics and Computer Science

BABAU Jean-Philippe *et al.*
Model Driven Engineering for Distributed Real-Time Embedded Systems 2009

BOULANGER Jean-Louis
Safety of Computer Architectures

MONMARCHE Nicolas *et al.*
Artificial Ants

PANETTO Hervé, BOUDJLIDA Nacer
Interoperability for Enterprise Software and Applications 2010

PASCHOS Vangelis Th
Combinatorial Optimization – 3-volume series
Concepts of Combinatorial Optimization – Volume 1
Problems and New Approaches – Volume 2
Applications of Combinatorial Optimization – Volume 3

SIGAUD Olivier *et al*.
Markov Decision Processes in Artificial Intelligence

SOLNON Christine
Ant Colony Optimization and Constraint Programming

AUBRUN Christophe, SIMON Daniel, SONG Ye-Qiong *et al.*
Co-design Approaches for Dependable Networked Control Systems

2009

FOURNIER Jean-Claude
Graph Theory and Applications

GUEDON Jeanpierre
The Mojette Transform / Theory and Applications

JARD Claude, ROUX Olivier
Communicating Embedded Systems / Software and Design

LECOUTRE Christophe
Constraint Networks / Targeting Simplicity for Techniques and Algorithms

2008

BANÂTRE Michel, MARRÓN Pedro José, OLLERO Hannibal, WOLITZ Adam
Cooperating Embedded Systems and Wireless Sensor Networks

MERZ Stephan, NAVET Nicolas
Modeling and Verification of Real-time Systems

PASCHOS Vangelis Th
Combinatorial Optimization and Theoretical Computer Science: Interfaces and Perspectives

WALDNER Jean-Baptiste
Nanocomputers and Swarm Intelligence

2007

BENHAMOU Frédéric, JUSSIEN Narendra, O'SULLIVAN Barry
Trends in Constraint Programming

JUSSIEN Narendra
A to Z of Sudoku

2006

BABAU Jean-Philippe *et al.*
From MDD Concepts to Experiments and Illustrations – DRES 2006

HABRIAS Henri, FRAPPIER Marc
Software Specification Methods

MURAT Cecile, PASCHOS Vangelis Th
Probabilistic Combinatorial Optimization on Graphs

PANETTO Hervé, BOUDJLIDA Nacer
Interoperability for Enterprise Software and Applications 2006 / IFAC-IFIP I-ESA'2006

2005

GÉRARD Sébastien *et al.*
Model Driven Engineering for Distributed Real Time Embedded Systems

PANETTO Hervé
Interoperability of Enterprise Software and Applications 2005