Introduction to modeling and simulation of communication networks

Cyril Nguyen Van Phu
COSYS-GRETTIA, Univ Gustave Eiffel, IFSTTAR, F-77454 Marne-la-Vallée, France
Email : cyril.nguyen-van-phu@univ-eiffel.fr

2020-2021

What is modeling ?


```
Modeling in its broadest sense is the cost-effective use of something in
place of something else for some purpose.  It allows us to use something
that is simpler, safer, or cheaper than reality instead of reality for some
purpose.  A model represents reality for the given purpose; the model
is an abstraction of reality in the sense that it cannot represent all
aspects of reality.  This allows us to deal with the world in a simplified
manner, avoiding the complexity, danger and irreversibility of reality.
```

Figure – "The nature of modeling" [1]

What is modeling ? (2)

Definition of Modeling

Precisely what do we mean by modeling? Modeling is a way of dealing with things or situations that are too "costly" to deal with directly (where "cost" is interpreted in the broadest sense). Any model is characterized by three essential attributes:

1) REFERENCE: It is of something (its "referent").

2) PURPOSE: It has some intended purpose with respect to its referent.

3) COST-EFFECTIVENESS: It is more cost-effective to use the model for this purpose than to use the referent itself.

Figure – "The nature of modeling" [1]

Introduction to the art and science of simulation [2]

"By a model we mean a representation of a group of objects or ideas in some form other than that of the entity itself. By a system we mean a group or collection of interrelated elements that cooperate to accomplish some stated objective."

Models in Science [3]

"Models raise questions in semantics (how, if at all, do models represent ?), ontology (what kind of things are models ?), epistemology (how do we learn and explain with models ?), and, of course, in other domains within philosophy of science."

Models in Science [3] : semantics

“Many scientific models are representational models : they represent a selected part or aspect of the world, which is the model's target system. “ How do model represent ?

- Scale models are ”down-sized or enlarged copies of their target systems“.
- Analogical models : ”two things are analogous if there are certain relevant similarities between them“
- ”Idealized models are models that involve a deliberate simplification or distortion of something complicated with the objective of making it more tractable or understandable.“
- ”A model of data (sometimes also “data model”) is a corrected, rectified, regimented, and in many instances idealized version of the data we gain from immediate observation“

Models in Science [3] : ontology

Ontology : What Are Models ?

- "Some models are physical objects. "
- Fictional objects and abstract objects : "The Bohr model of the atom, a frictionless pendulum, or an isolated population, for instance, are in the scientist's mind rather than in the laboratory and they do not have to be physically realized and experimented upon to serve as models. "
- "Descriptions and equations"

Models in Science [3] : epistemology

The Cognitive Functions of Models

- ”learning about models“
- ”learning about target systems“
- ”Explaining with models“

Modeling modeling modeling [4]

”we emphasize the major importance of the intentional nature of modeling and introduce intention as a first-class property of the representation relation between two things used in a modeling process.“

Model driven architecture : Principles and practice [5]

complex environment and complex target infrastructures. Two ideas :

- service oriented architectures : viewing ”enterprise solutions as federations of services connected via well-specified contracts that define their service interfaces.“
- software product lines

Model driven architecture : Principles and practice [5]

Example of model with the purpose of building sofware architectures : Service oriented architectures : ”Many organizations now express their solutions in terms of services and their interconnections. A number of important technologies have been defined to support an SOA approach, most notably when the services are distributed across multiple machines and connected by the Internet. These web service approaches rely on intra-service communication protocols such as the Simple Object Access Protocol (SOAP), allow the web service interfaces (expressed in the Web Services Definition Language-WSDL) to be registered in public directories and searched in Universal Description, Discovery and Integration (UDDI) repositories, and share information in documents defined in the eXtensible Markup Language (XML) and described in standard schemas.“

Model driven architecture : Principles and practice [5]

Software product lines : "Organizations are also beginning to recognize that there frequently is a great deal of commonality in the systems they develop. Recurring approaches are seen at every level of an enterprise software project, from having standard domain models that capture core business processes and domain concepts, to the way in which developers implement specific solutions to realize designs in code. A great deal of efficiency can be gained if patterns can be defined by more skilled developers and propagated across the IT organization. Recognizing this, many organizations are moving to- ward a software product line view of their development in which planned reuse of assets is supported, and an increasing level of automation can be used to realize solutions for large parts of the systems being developed [2, 3]."

On the modeling and analysis of computer networks [6]

The fact is, there are a number of ways in which you can go about doing network system performance analysis. In order of increasing ugliness, they are as follows:

1) Conduct a mathematical analysis which yields explicit performance expressions.
2) Conduct a mathematical analysis which yields an algorithmic or numerical evaluation procedure.
3) Write and run a simulation.
4) Build the system and then measure its performance!

On understanding

“The highest activity a human being can attain is learning for understanding, because to understand is to be free. ” Baruch Spinoza, Ethics.

On models

"All models are wrong, some are useful." George Box.

Example : mobility model

definition of the travel distance :

$$d = vt \tag{1}$$

- pros : the model is useful and simple.
- cons : the need to know v and t. In quantum physics, Heinseberg inequality states that $\Delta E \Delta t \geq \frac{\hbar}{2}$.

Open Systems Interconnection Model (1984)

OSI Model			
	Data unit	**Layer**	**Function**
Host layers	Data	7. Application	Network process to application
		6. Presentation	Data representation, encryption and decryption, convert machine dependent data to machine independent data
		5. Session	Interhost communication, managing sessions between applications
	Segments	4. Transport	Reliable delivery of segments between points on a network.
Media layers	Packet/Datagram	3. Network	Addressing, routing and (not necessarily reliable) delivery of datagrams between points on a network.
	Bit/Frame	2. Data link	A reliable direct point-to-point data connection.
	Bit	1. Physical	A (not necessarily reliable) direct point-to-point data connection.

Useful and rigorous model but TCP/IP model is more pliant and widely spread.

Principles of modeling [7]

In this paper the author lists various modeling principles from different sources.

What is simulation ? [8]

Compilation of definitions of simulation

by

A. Alan B. Pritsker
Pritsker & Associates, Inc.
and *Purdue University*
West Lafayette, Indiana 47907

A. ALAN B. PRITSKER is a professor of industrial engineering at Purdue University and president of Pritsker & Associates, Inc. Over the years he has, he says, been asked many times for a precise definition of simulation, *and in response he normally gave those included in his own books. Recently, however, he decided to compare different definitions and compiled the list that follows for the use of his students and associates. He reports that it has indeed turned out to be useful to them and hopes that it will also be useful to our readers. We think it will and invite you to send in your favorite definitions. We'll collect them and at some future time publish an addendum to the present set.*

Simulation modeling assumes that we can describe a system in terms acceptable to a computing system. In this regard, a key concept is that of a *system state description*. If a system can be characterized by a set of variables, with each combination of variable values representing a unique state or condition of the system, then manipulation of the variable values simulates movement of the system from state to state. This is precisely what simulation is: *the representation of the dynamic behavior of the system by moving it from state to state in accordance with well-defined operating rules.*

PRITSKER, A.A.B.   PEGDEN, C.D.
Introduction to Simulation and SLAM
Halsted Press   New York   1979   p. 6
Systems Publishing Corporation   West Lafayette, Indiana   1979   p. 6

Simulation denotes a computer-based numerical technique for the experimental study of a stochastic or deterministic process over time.

SCHRUBEN, L.W.   MARGOLIN, B.M.
Random Number Assignment in Statistically Designed Simulation and Distribution Sampling Experiments
Journal of the American Statistical Association
vol. 73   no. 363   September 1978   p. 504

Simulation is thus viewed as the operation — from input through output — of a computer program. The program is a process model of the operation of some real occurrence and is based on, and perhaps indistinguishable from, a theory concerning it. A computer model and a simulation are tools in attempting to understand the nature of the laws governing behavior; they help in prediction and explanation, which are the central elements in science.

LEHMAN, R.S.
Computer Simulation and Modeling: an Introduction
New York   John Wiley & Sons   1977   p. 8

Simulation is the technique of constructing and running a model of a real system in order to study the behavior of that system, without disrupting the environment of the real system.

BOBILLIER, P.A.   KAHAN, B.C.   PROBST, A.R.
Simulation with GPSS and GPSS V
Prentice-Hall   Englewood Cliffs, New Jersey
1976   p. 6

The phrase "modeling and simulation" designates the complex of activities associated with constructing models of real world systems and simulating them on a computer. In particular, *modelling* deals primarily with the relationships between real systems and models; *simulation* refers primarily to the relationships between computers and models.

ZEIGLER, B.P.
Theory of Modelling and Simulation
John Wiley & Sons   New York   1976   p. 3

We can therefore define system simulation as the technique of solving problems by the observation of the performance, over time, of a dynamic model of the system.

GORDON, G.
The Application of GPSS V to Discrete System Simulation
Prentice-Hall   Englewood Cliffs, New Jersey
1975   p. 7

Simulation is the process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of the system.

SHANNON, R.E.
Systems Simulation: the Art and the Science
Prentice-Hall   Englewood Cliffs, New Jersey
1975   p. 2

We define *simulation* in a *narrow* sense as experimenting with an (abstract) model over time, this experimentation involving the sampling of values of stochastic variables from their distributions. Therefore this simulation is called *stochastic* simulation.

KLEIJNEN, J.P.C.
Statistical Techniques in Simulation: Part I
Marcel Dekker, Inc.   New York   1974   p. 14

Simulation is the establishment of a mathematical-logical model of a system and the experimental manipulation of it on a digital computer.

PRITSKER, A.A.B.
The GASP IV Simulation Language
John Wiley & Sons   New York   1974   p. 1

System simulation means the act of representing a system by a symbolic model that can be manipulated easily and that produces numerical results. (p. 14)

Discrete event digital simulation concerns the modeling on a digital computer of a system in which state changes can be represented by a collection of discrete events. (p. 23)

FISHMAN, G.S.
Concepts and Methods in Discrete Event Digital Simulation
John Wiley & Sons   New York   1973   pp. 14, 23

Simulation is the use of a model (not necessarily a computer model) to carry out experiments designed to reveal certain characteristics of the model and by implication of the idea, system, or situation modeled.

McLEOD, J.
Simulation Today—from Fuzz to Fact
Simulation   March 1973   center section

Simulation is . . . seen as . . . a process—the operation of a model—but a process that is in some sense a copy of or parallel to a real process, the latter being the real-world process that is of interest in the theoretical context. Stated in another way, simulation is the model in operation.

SCHULTZ, R.L.   SULLIVAN, E.M.
Developments in Simulation in Social and Administrative Science
In H. Guetzkow, P. Kotler, and R.L. Schultz, editors, *Simulation in Social and Administrative Science: Overviews and Case-Examples* (Prentice-Hall, Englewood Cliffs, New Jersey, 1972), p. 7

Modeling should refer to the gathering and structuring of data in such a way that the values of the parameters, the initial values of the variables, and their interrelations are formalized. The models may be conceptual, physical, mathematical, or computerized—or a concurrent or progressive combination of these.

The term *simulation*, strictly speaking, should be reserved to mean the use of a model to carry out "experiments" specifically designed to study selected aspects of the *simuland*, i.e., the real-world or hypothesized system that has been modeled.

McLEOD, J.
Simulation Today—and Yesterday
Simulation   May 1972   p. 3

We shall define simulation to be the action of performing experiments on a model of a given system.

SCHMIDT, J.W.   TAYLOR, R.E.
Simulation and Analysis of Industrial Systems
Richard D. Irwin, Inc.   Homewood, Illinois
1970   p. 4

We therefore define system simulation as the technique of solving problems by following the changes over time of a dynamic model of a system.

GORDON, G.
System Simulation
Prentice-Hall   Englewood Cliffs, New Jersey
1969   p. 17

The term simulation refers to " . . . the exercise of flexible imitation of processes and outcomes for the purpose of clarifying or explaining the underlying mechanisms involved."

ABELSON, R.P.
Simulation of Social Behavior
In G. Lindzey and E. Aronson, editors, *The Handbook of Social Psychology. Vol. 2: Research Methods* (Addison-Wesley, Reading Massachusetts, 1968), 2nd edition, p. 275

During recent years the word "simulation" has been adopted from colloquial language to serve as a technical term. Because of this, the word has been used in different and often conflicting senses. The existing confusion is due not so much to carelessness on the part of scientific researchers as it is to the nature of the word "simulation" itself. Its history and usage in non-technical contexts have made it a rather poor choice for technical use since dictionaries permit exceedingly broad and vague definitions for the word. These definitions hinge on such potentially divergent concepts as *representing*, *imitating*, *shamming*, *counterfeiting*, *appearing*, and so on. As a result, the word has long been used to stand for an assortment of different ideas, some of them quite imprecisely defined. This ambiguity has been carried over to the scientific and technical fields to the extent that other writers hold that a model is not itself a simulation, but rather a thing used in simulation. Similarly, in some situations where both a model and a computer routine are used, some writers may say that the computer routine "simulates the model," but other writers maintain that the routine performs a function entirely different from that of simulating the model. Some people speak of simulation as if it were a thing, some as if it were an activity, some as if it were a relationship, and so on. In order for a word to be useful as a technical term in scientific research, the number of ideas it can stand for must be quite small, and the nature of those ideas must be sharply defined. (p. 1)

. . . given a system and a model of that system, *simulation is the use of the model to produce chronologically a state history of the model, which is regarded as a state history of the modeled system.* (p.6)

EVANS, G.W., II WALLACE, G. SUTHERLAND, G.L.
Simulation Using Digital Computers
Prentice-Hall Englewood Cliffs, New Jersey
1967

Simulation is a term commonly applied to the use of models to study systems. We consider a narrower definition of simulation—the use of numerical models for the study of systems. Simulation is the use of a numerical model to study the behavior of a system as it operates over time.

KIVIAT, P.J.
Digital Computer Simulation: Modeling Concepts
Memorandum RM-5378-PR The RAND Corporation
1967 p. 4

Simulation is a numerical technique for conducting experiments on a digital computer, which involves certain types of mathematical and logical models that describe the behavior of a business or economic system (or some component thereof) over extended periods of time.

NAYLOR, T.H. *et al.*
Computer Simulation Techniques
John Wiley & Sons New York 1966 p. 3

Simulations, symbolic models in which none of the physical characteristics of the modeled is reproduced in the model itself, yet in which the symbols are not manipulated entirely by a well-formed discipline in order to arrive either at a particular numerical value or at an analytic solution.

SAYRE, K.M. CROSSON, F.H., editors
The Modeling of Mind: Computers and Intelligence
Simon and Schuster New York 1963

Simulation is the process of determining the sampling distribution of a highly irregular and intricately defined statistic.

TOCHER, K.D.
The Art of Simulation
D. Van Nostrand Co. Princeton, New Jersey
1963 pp. 3-4 As paraphrased by:
WYMAN, F.P.
Simulation Modeling: a Guide to Using SIMSCRIPT
John Wiley & Sons New York 1970 p.2

To simulate means to duplicate the essence of the system or activity without actually attaining reality itself.

MORGENTHALER, G.W.
The Theory and Application of Simulation in Operations Research
In R.L. Ackoff, editor, *Progress in Operations Research* (John Wiley & Sons, New York, 1961), vol. 1, p. 367

SCS Regional Council election results

SOCIETY FOR COMPUTER SIMULATION
MIDWESTERN REGIONAL COUNCIL
STEERING COMMITTEE 1979-80

CHAIRMAN

William O. Grierson
Bendix Research Laboratories
Bendix Center
20800 Civic Center Drive
Southfield, MI 48076
(313) 352-7731

VICE CHAIRMAN

Joseph J. Marous
Wright Patterson AFB ASD/ADSS
Dayton, OH 45433
(513) 255-6196

SECRETARY

Donald J. Ewing, Jr.
608 Pierce Street
Maumee, OH 43537

TREASURER

Andrew Frank, Director
Data Acquisition & Simulation Laboratory
University of Wisconsin
1500 Johnson Drive
Madison, WI 53706
(608) 262-1234

DIRECTOR

Eugene Henry
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556
(219) 284-6408

DIRECTOR

Robert D. Gustafson
Simulation Specialists, Inc.
609 W. Stratford Place
Chicago, IL 60657
(312) 528-7433

MEMBERS-AT-LARGE

John R. Carter
5166 Scarsdale Drive
Kettering, OH 45440
(513) 435-5417

MEMBERS-AT-LARGE

Donald Chandler
Sales Manager
Applied Dynamics International
3800 Stone School Road
Ann Arbor, MI 48104
(313) 973-1300

David Filmer
Department of Biological Science
Purdue University
Lafayette, Indiana 47507

Everett Keener
Associate Research Consultant
Research Laboratory
U.S. Steel Corporation
125 Jamison Lane
Monroeville, PA 15146

Robert Schwing
c/o Eugene Henry
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556
(219) 284-6408

What is simulation ?(2)

Modeling and Simulation: Definitions

1. Definitions of "simulation" on the Internet

2. M&S Definitions - from Defense-related Sources

Modeling and Simulation (M&S):
"The use of models and simulations, either statistically or over time, to develop data as a basis for making managerial or technical decisions. This includes, but is not limited to, emulators, prototypes, simulators, and stimulators." DOD M&S VV&A, DoDI

Modeling and Simulation (M&S)
The use of models, including emulators, prototypes, simulators, and stimulators, either statically or over time, to develop data as a basis for making managerial or technical decisions. The terms "modeling" and "simulation" are often used interchangeably. DMSO M&S Glossary

Simulation:
"A simulation is a method for implementing a model. It is the process of conducting experiments with a model for the purpose of understanding the behavior of the system modeled under selected conditions or of evaluating various strategies for the operation of the system within the limits imposed by developmental or operational criteria. Simulation may include the use of analog or digital devices, laboratory models, or 'testbed' sites. Simulations are usually programmed for solution on a computer; however, in the broadest sense, military exercises, and wargames are also simulations." DAU

Simulation:
"A method for implementing a model over time. Also a technique for testing, analysis, or training in which real-world systems are used, or where real-world and conceptual systems are reproduced by a model (reference b: 'DoD modeling and Simulation (M&S) Management,' January 4, 1994) DOD M&S VV&A

Simulation:
"*the exercising of a model over time. A simulation may be:*
- *'live' - exercising real people and/or real equipments in the real world (e.g. a live trial or exercise)*
- *'virtual' - exercising real people and virtual people/equipments, possibly in a virtual world*
- *'constructive' - exercising virtual people and/or equipments, usually in a virtual world.*"
  UK-SEMS

Simulation:
"A method for the implementation of a model over time. (DoD 5000.59-M)" DoD A M&S MP

Simulation:
"A simulation is the implementation of a model over time. Human-in-the-loop (HITL) simulations are commonly referred to as "simulators" DND/CF

Simulation:
"The execution over time of models representing the attributes of one or more entities or processes. Human-in-the-Loop simulations, also known as simulators, are a special class of simulations." NATO M&S Master Plan version 1.0 1998, August 7

3. M&S Definitions - from Civilian Sources

- **An early compilation** of definitions of simulation was done by Pritsker: Pritsker, A.A.B. (1979). Compilation of Definitions of Simulation. Simulation, 33: 2 (Aug. 1979), 61-63.

- From a systemic point of view simulation can be used to find the values of output, input, or state variables of a system; provided that the values of the two other types of variables are known."

4. Some Suggested Definitions of Simulation

Experience aspect of simulation

- **Definition:** Simulation is use of a representation (a model) of reality to provide experience under controlled conditions (1) for training to gain/enhance three types of skills; or (2) for entertainment.

- **Aim**:
  - Simulation is used to enhance
    (1) motor skills (virtual simulation),
    (2) decision making and/or communications skills
    (constructive simulation, serious games, gaming simulation), and
    (3) operational skills by getting real-life-like experience in a controlled environment
    (live simulation).
  - Simulation is used for entertainment (simulation games)

- **Application categories**: Simulation is used for training, education, and entertainment.

Experiment aspect of simulation

- **Definition:** Simulation is goal-directed experimentation with dynamic models.
- **Application categories**: Simulation is used for understanding, decision support, and education.

A comprehensive perception of simulation

- Simulation offers a very rich paradigm to perform experiments with dynamic models and to provide experience either for entertainment or for training to develop/enhance three types of skill, i.e., motor skills, decision-making skills, or operational skills. For the last three categories, associated three types of simulation are virtual, constructive and live simulations, respectively.

Model-oriented activity perception of simulation provides several additional practical and methodological benefits.

Introduction to the art and science of simulation [2]

"We will define simulation as the process of designing a model of a real system and conducting experiments with this model for the purpose of understanding the behavior of the system and/or evaluating various strategies for the operation of the system. Thus it is critical that the model be designed in such a way that the model behavior mimics the response behavior of the real system to events that take place over time."

Traffic Modeling For Telecommunications Networks [9]

”Because executing a simulation is analogous to conducting an experiment involving randomness, simulation outputs must be treated as random observations. In a similar vein, because a model is viewed as a faithful representation of the target system, instrumentation is required in order to collect statistics and formulate performance predictions. The success of a simulation study hinges on identifying appropriate performance metrics and then devising a strategy for exploring the ensuing performance response surface.“

Group talk

- How would you model the dynamics of a queue, and queuing process ? Which parts would be required for this modeling ?
- Considering such a queue model, what would it be useful for ? (and in the context of networking communication simulation) ?

Queueing theory

Queueing theory is used to model concurrent access to shared resources. We want :

- a model for the arrivals of the clients
- a model for the service of the clients
- a model for the dynamics of the queueing process
- with indicators of performances such as the mean waiting time, mean number of customers in the queue...

Kendall's notation

The basic queue is the M/M/1, it is generalized into the notation A/B/C/K/Z where :

- A : inter-arrival time distribution (arrivals)
- B : service time distribution
- C : the number of servers
- K : system capacity
- Z : service discipline

A and B can be M (memoryless), D (deterministic, constant), G (general distribution).
Z can be FIFO, LIFO, RS (random service) , PS (processor sharing), priority
Default values : K=infinite and Z=FIFO

Queueing theory ?

Queueing theory adress the problems where the resources provided are limited.
Simple formulas are not always known, so we use :

- Approximations/limits
- Simulations

Simple models like M/M/1, M/M/1/N, M/M/k/k are very useful in checking simulations and understanding convergence time !
Some cases like the M/M/1 queue are solved and we have the mathematical solution (for the steady case analysis) but it is not the general case. Hence, simulation of queues is widely used.

Example 1 : Guaranteed Time Slots (GTS) of Medium Access Control (communication protocol IEEE 802.15.4)

- We consider a fixed number of time slots which are a limited resource and which can be reserved : Guaranteed Time Slots (GTS)
- Clients are nodes who want to reserve the time slots in advance ; arrivals are GTS requests.
- Metrics outputs should give average throughput for the communication flow of a node.

M/D/1 queue with mathematical solution :

- GTS requests are Poisson arrivals
- Service duration is deterministic (time slots of fixed size)
- 1 coordinator
- FIFO queueing discipline

Example 2 : Output Port on an IP router [10]

- General random IP packet arrivals
- Constant bit rate output port, but random packet length leads to General random service times
- Each port acts as a server (so one server)
- Large but finite output buffer size
- Assume FIFO and infinite population

Model : G/G/1/N queueing system

Example 3 : Classic Telephone Trunk [10]

- Number of servers is the number of TDM (time division multiplexing) timeslots on a link
- Service times = call duration are exponentially distributed
- No waiting. Only as many users as time slots. Calls arriving when all servers were in use were blocked and given a "busy signal"

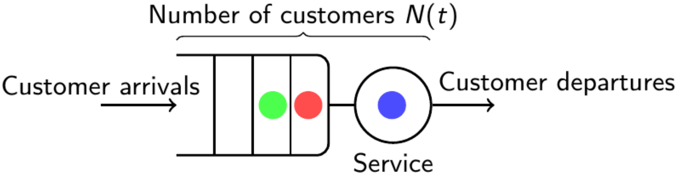
Model : M/M/m/m *M* denotes either "Markov" or "Memoryless" and implies exponential distribution *m* the numbers of servers and capacity in the system Key performance parameter : blocking probabilities

Example 4 : Access to a cell base station from handset [10]

- Fixed number of available access slots (frequency and time slots) “servers”
- General random arrival process. What would influence this ?
- General random service time. What would influence this ?

Model : G/G/m/m Where m is the total number of available timeslot x frequencies (servers) and there is only m “spaces” in the system for customers. Interested in call dropping and blocking statistics

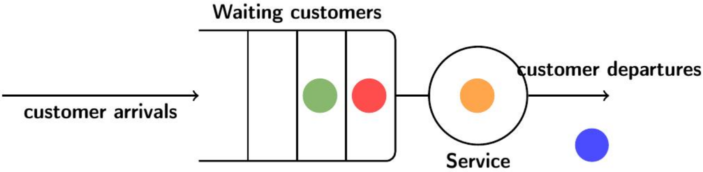
M/M/1 Queue model



- Poisson arrivals
- Exponential service times
- 1 server
- infinite queue length (infinite buffer capacity)
- FIFO queueing discipline

Prerequisites : probability theory

M/M/1 Queue model : example



"Consider a queue with one server and an infinite waiting buffer. Clients need a service that can be provided by the server, and if the server is empty they can wait in the buffer until the server is free. Here, when the first client arrives – the blue one – the server is free so the blue client starts to be served immediately. A second orange colored client arrives before the service of the blue client has finished, so that the orange client is stored in first position in the buffer. A third red colored client arrives and the server is still busy serving the blue client. So the red client waits in second position in the buffer. Again a new green colored client arrives and is stored in the next position in the buffer. Then in this example, when the blue client is no longer being served it leaves the system and releases the server. The first client in the waiting buffer – the orange one – then starts being served and all the other clients move forward one position in the buffer."

Arrival Poisson process : definitions and notations

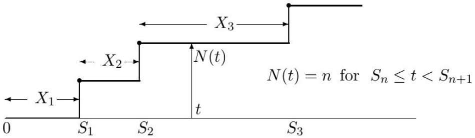


Figure – Notations. Figure taken from [11].

This chapter on the Poisson process is based on the material given in [11]. Arrival process characterized by either :

- inter-arrival times, epoch times and their relations : $\{X_i; i \geq 1\}$, $X_i = Si - S_{i-1}$, $S_n = \sum_{i=1}^{n} X_i$ or
- counting process $\{N(t); t \geq 0\}$ with $\{S_n \leq t\} = \{N(t) \geq n\}$.

Poisson process definition

Definition

A renewal process is an arrival process for which the sequence of interarrival times is a sequence of positive Independent and Identically Distributed (IID) random variables.

Definition

A Poisson process is a renewal process in which the interarrival times have an exponential cumulative distribution function (CDF) ; i.e., for some real $\lambda > 0$, each X_i has the CDF $F_X(x) = P(X \leq x) = 1 - exp(-\lambda x)$ which is equivalent to say that the density $f_X(x) = \lambda exp(-\lambda x)$, for $x \geq 0$.

λ is the rate of the process. λt is the expected number of arrivals in interval t (in average) : see the stationnary increment property.

Memoryless property

Definition

A random variable X possesses the memoryless property if X is a positive rv (i.e., $Pr(X > 0) = 1$) for which :
$P(X > x + t) = P(X > x)P(X > t)$ for all $x, t \geq 0$.

The idea of this definition is following :

$$\begin{aligned} P(X > x + t) &= P(X > x + t, X > t) \\ &= P(X > x + t | X > t) P(X > t) \text{ by Bayes theorem} \\ &= P(X > x) P(X > t) \end{aligned} \tag{2}$$

Check that a Poisson process is memoryless.

A Poisson process is a probabilistic replica of itself starting at time 0

Theorem

For a Poisson process of rate λ, and any given $t > 0$, the length of the interval from t until the first arrival after t is a nonnegative random variable Z with the CDF $F_Z(z) = 1 - exp(-\lambda z)$, for $z \geq 0$. This random variable is independent of both $N(t)$ and of the $N(t)$ arrival epochs before time t. It is also independent of the set of random variables $N(\tau); \tau \leq t$.

Proof of Theorem 1 : inference from the Poisson process definition and the memoryless property

“The basic idea behind this theorem is that Z, conditional on the epoch τ of the last arrival before t, is simply the remaining time until the next arrival. Since the interarrival time starting at τ is exponential and thus memoryless, Z is independent of $\tau \leq t$, and of all earlier arrivals.”

Proof of Theorem 1 : figure and notations

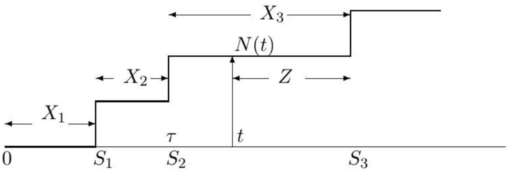


Figure 2.3: Given $N(t) = 2$, and $S_2 = \tau$, X_3 is equal to $Z + (t - \tau)$. Also, the event $\{N(t){=}2,\ S_2{=}\tau\}$ is the same as the event $\{S_2{=}\tau,\ X_3{>}t{-}\tau\}$.

Proof of Theorem 1

Conditional on $N(t) = n$ and $S_n = \tau$, the first arrival after t is the first arrival after the arrival at S_n , i.e., $Z = z$ corresponds to $X_{n+1} = z + (t - \tau)$.

$$\begin{aligned} P(Z > z | N(t) = n, S_n = \tau) = \\ P(X_{n+1} > z + t - \tau | N(t) = n, S_n = \tau) = \\ P(X_{n+1} > z + t - \tau | X_{n+1} > t - \tau, S_n = \tau) \end{aligned} \quad (3)$$

Because X_{n+1} is indepedent of S_n, $\{X_{n+1} > t - \tau, S_n = \tau\} = \{X_{n+1} > t - \tau\}$. Then,

$$\begin{aligned} P(Z > z | N(t) = n, S_n = \tau) = \\ P(X_{n+1} > z + t - \tau | X_{n+1} > t - \tau) \end{aligned} \quad (4)$$

Proof of Theorem 1

Because of the memoryless property :

$$P(Z > z|N(t) = n, S_n = \tau) = P(X_{n+1} > z) = exp(-\lambda z) \quad (5)$$

As all the S_i are iid random variables, it is also true if we condition equation (5) not only on S_n but also on $S_1, ..., S_{n-1}$. This is equivalent to condition on $N(\tau)$ for all $\tau < t$ such that :

$$P(Z > z|\{N(\tau), 0 \leq \tau < t\}) = exp(-\lambda z) \quad (6)$$

We have given the arguments for $N(t) = n$, with $n > 1$. Concerning the initial case $n = 1$ the arguments are basically the same.

Stationary increment property

Definition

A counting process $\{N(t); t > 0\}$ has the stationary increment property if $N(t') - N(t)$ has the same CDF as $N(t' - t)$ for every $t' > t > 0$.

This means that $N(t') - N(t)$ depends only on the length of the interval $(t' - t)$. A Poisson process has the stationary increment property because it is a replica of itself starting at time 0 : Let us examine this property of the Poisson process.

A Poisson process is a replica of itself starting at time 0 : the stationary increment property

"For $m \geq 2$, let Z_m be the time from the $(m-1)$st arrival after t to the mth arrival epoch after t. Thus, given $N(t) = n$ and $S_n = \tau$, we see that $Z_m = X_{m+n}$ for $m \geq 2$. It follows that, conditional on $N(t) = n$ and $S_n = \tau$, $Z_1, Z_2, ..,$ are IID exponentially distributed random variables. This is because the X_i are IID random variables, by definition. Since this is independent of $N(t)$ and S_n, it follows that $Z_1, Z_2, ..,$ are unconditionally IID random variables and also independent of $N(t)$ and S_n. It should also be clear that $Z_1, Z_2, ..,$ are independent of $\{N(\tau); 0 < \tau \leq t\}$. Thus, a Poisson process starting at an arbitrary time $t > 0$ is a probabilistic replica of the process starting at time 0 : the time until the first arrival after t is an exponentially distributed rv with parameter λ, and all subsequent interarrival times are independent of this first arrival epoch and of each other, and all have the same exponential distribution."

Stationary independent increment property

A Poisson process $\{N(t); t > 0\}$ has the independent increment property :
for every integer $k > 0$, and every k-tuple of times $0 < t_1 < t_2 < ... < t_k$, the k-tuple of random variables $N(t_1)$, $(N(t2) - N(t_1))$,...,$(N(t_k) - N(t_{k-1}))$ are statistically independent.

Joint density of $S_1 ... S_n$

Theorem

Let $X_1, X_2, ..., X_n$ be IID random variables with the density $f_X(x) = \lambda exp(-\lambda x)$ for $x \geq 0$. Let $S_n = X_1 + ... + X_n$ for each $n \geq 1$. Then for each $n \geq 2$:

$$f_{S_1,...S_n}(s_1, ..., s_n) = \lambda^n exp(-\lambda s_n)$$

for $0 \leq s_1 \leq s_2 \leq ... \leq s_n$.

Proof by recurrence : Joint density of $S_1 ... S_n$

The theorem holds for $n = 2$. Let us assume that it holds for n. Then :

$$f_{S_1,\ldots S_n,S_{n+1}}(s_1, \ldots, s_n, s_{n+1}) = f_{S_{n+1}|S_1,\ldots S_n}(s_{n+1}|s_1, \ldots, s_n) f_{S_1,\ldots S_n}(s_1, \ldots, s_n) = f_{S_{n+1}|S_1,\ldots S_n}(s_{N+1}|s_1, \ldots, s_n)\lambda^n exp(-\lambda s_n) \quad (7)$$

As $S_{n+1} = S_n + X_{n+1}$ and X_{n+1} is independent of $S_1, \ldots S_n$:
$f_{S_{n+1}|S_1,\ldots S_n}(s_{N+1}|s_1, \ldots, s_n) = \lambda exp(-\lambda(s_{n+1} - s_n))$

Probability density of S_n : Erlang density

$$f_{S_n}(t) = \frac{\lambda^n t^{n-1} exp(-\lambda t)}{(n-1)!} \tag{8}$$

Démonstration.

We integrate $f_{S_1,\ldots S_n}(s_1, \ldots, s_n)$ and compute the marginal probability density : $f_{S_n}(s_n) = \int_{s_1} \ldots \int_{s_{n-1}} f_{S_1,\ldots S_n}(s_1, \ldots, s_n)$ □

Probability mass function for $N(t)$

Theorem

For a Poisson process of rate λ , and for any $t > 0$, the Probability Mass Function (PMF) for $N(t)$, i.e., the number of arrivals in $]0, t]$, is given by the Poisson PMF :

$$p_{N(t)}(n) = \frac{(\lambda t)^n exp(-\lambda t)}{n!} \tag{9}$$

Proof : Probability mass function for $N(t)$

We compute $P(t < S_{n+1} \leq t + \delta)$. By approximating the density $f_{S_{n+1}}$ as constant in the interval of length δ :

$$P(t < S_{n+1} \leq t + \delta) = \int_t^{t+\delta} f_{S_{n+1}}(\tau)d\tau = f_{S_{n+1}}(\delta + o(\delta)) \quad (10)$$

where $o(\delta)$ tends to zero when δ tends to zero.

Proof : Probability mass function for $N(t)$

We compute $P(t < S_{n+1} \leq t + \delta)$ in another way :

$$P(t < S_{n+1} \leq t + \delta) = p_N(n)(\lambda\delta + o(\delta)) + o(\delta) \quad (11)$$

This probability is the sum of two events (logical disjunction, operator "or") :

- The term $p_N(n)(\lambda\delta + o(\delta))$ is because of the independent increment property. We compute $F_{S_1 \leq \delta} = \int_0^\delta \lambda exp(-\lambda t) = 1 - exp(-\lambda\delta) = \lambda\delta + o(\delta)$ which represents the probability that there is one arrival in $]t, t + \delta]$. $p_N(n)$ is the probability that there are N arrivals in $]0, t]$. The product of the two is the joint probability that there are N arrivals in $]0, t]$ and one arrival in $]t, t + \delta]$.
- The term $o(\delta) = p_N(n)o(\delta)$ represents the probability that there are N arrivals in $]0, t]$ and more than one arrival in $]t, t + \delta]$.

Proof : Probability mass function for $N(t)$

We can conclude this proof by equaling the two manners of having calculated $P(t < S_{n+1} \leq t + \delta)$ and tending δ to zero :

$$f_{S_{n+1}}(\delta + o(\delta)) = p_N(n)(\lambda\delta + o(\delta)) + o(\delta)$$
$$p_N(n) = (1/\lambda) f_{S_{n+1}} = (1/\lambda)\frac{\lambda^{n+1} t^n exp(-\lambda t)}{n!} = \frac{(\lambda t)^n exp(-\lambda t)}{n!} \tag{12}$$

Number of arrivals in a very small interval

Because of the independent increment property, we can use the formula of the probability mass function $p_N(n)$ in an interval of length δ. Then we use the Taylor expansion of $exp(x)$ as x tends to zero : $exp(x) = \sum_{n=0}^{+\infty} \frac{x^n}{n!} = 1 + x + o(x)$:

$$\begin{aligned} P(N(t+\delta) - N(t) = 0) = exp(-\lambda\delta) = 1 - \lambda\delta + o(\delta) \\ P(N(t+\delta) - N(t) = 1) = (\lambda\delta)exp(-\lambda\delta) = \lambda\delta + o(\delta) \\ P(N(t+\delta) - N(t) > 1) = o(\delta) \end{aligned} \tag{13}$$

M/M/1 queue. Material taken from Institut Mines Télécom opencourse [12]

$N(t)$ is the number of customers in the queue at time t

The evolution of $N(t)$ depending on differents events in $[t, t + \Delta t]$:

- one arrival
- one service
- no arrival, no service

Probabilities of the events

Let us recall that :

$$P(N(t+\delta) - N(t) = 1) = (\lambda\delta)exp(-\lambda\delta) = \lambda\delta + o(\delta)$$
$$P(N(t+\delta) - N(t) > 1) = o(\delta) \tag{14}$$

Such that :

$$P('onearrival') = \lambda \Delta t + o(\Delta t) \tag{15}$$

Moreover the probability of a service during Δt is :

$$P('oneservice') = \mu \Delta t + o(\Delta t) \tag{16}$$

$$P('noevent') = 1 - (\lambda + \mu)\Delta t + o(\Delta t) \tag{17}$$

Evolution of $N(t)$

Using the marginal conditional distribution :

$$
\begin{aligned}
P(N(t+\Delta t) = i) = & \qquad (18) \\
P(N(t+\Delta t) = i, N(t) = i-1) + & \qquad (19) \\
P(N(t+\Delta t) = i, N(t) = i) + & \qquad (20) \\
P(N(t+\Delta t) = i, N(t) = i+1) & \qquad (21)
\end{aligned}
$$

Evolution of $N(t)$

Using Bayes theorem :

$$P(N(t+\Delta t)=i) = \tag{22}$$
$$P(N(t+\Delta t)=i|N(t)=i-1)P(N(t)=i-1)+ \tag{23}$$
$$P(N(t+\Delta t)=i|N(t)=i)P(N(t)=i)+ \tag{24}$$
$$P(N(t+\Delta t)=i|N(t)=i+1)P(N(t)=i+1) \tag{25}$$

Evolution of $N(t)$

Let us denote $\pi_i(t) = P(N(t) = i)$:

$$\pi_i(t + \Delta t) = (\lambda \Delta t + o(\Delta t))\pi_{i-1}(t) + \tag{26}$$
$$(1 - (\lambda + \mu)\Delta t + o(\Delta t))\pi_i(t) + \tag{27}$$
$$(\mu \Delta t + o(\Delta t))\pi_{i+1}(t) \tag{28}$$

Evolution of $N(t)$

Let us denote $\pi_i(t) = P(N(t) = i)$:

$$\pi_i(t + \Delta t) = \lambda \Delta t \pi_{i-1}(t) + \quad (29)$$
$$(1 - (\lambda + \mu)\Delta t)\pi_i(t) + \quad (30)$$
$$\mu \Delta t \pi_{i+1}(t) + o(\Delta t) \quad (31)$$

Université Gustave Eiffel

Evolution of $N(t)$ for $i = 0$

$$\pi_0(t + \Delta t) = (1 - \lambda \Delta t)\pi_0(t) + \tag{32}$$

$$\mu \Delta t \pi_1(t) + o(\Delta t) \tag{33}$$

Chapman Kolmogorov equations

For $i \geq 1$

$$\pi_i(t+\Delta t) = \lambda \Delta t \pi_{i-1}(t) + \tag{34}$$
$$(1-(\lambda+\mu)\Delta t)\pi_i(t) + \tag{35}$$
$$\mu \Delta t \pi_{i+1}(t) + o(\Delta t) \tag{36}$$

and for $i = 0$:

$$\pi_0(t+\Delta t) = (1-\lambda \Delta t)\pi_0(t) + \tag{37}$$
$$\mu \Delta t \pi_1(t) + o(\Delta t) \tag{38}$$

Steady state analysis

We can write by substracting $\pi_i(t)$ (respectively $\pi_0(t)$) on both sides : For $i \geq 1$

$$\pi_i(t+\Delta t) - \pi_i(t) = \lambda \Delta t \pi_{i-1}(t) - \quad (39)$$
$$(\lambda + \mu)\Delta t \pi_i(t) + \quad (40)$$
$$\mu \Delta t \pi_{i+1}(t) + o(\Delta t) \quad (41)$$

and for $i = 0$:

$$\pi_0(t+\Delta t) - \pi_0(t) = -\lambda \Delta t \pi_0(t) + \quad (42)$$
$$\mu \Delta t \pi_1(t) + o(\Delta t) \quad (43)$$

Steady state analysis

Dividing by Δt and making Δt tend to 0 : For $i \geq 1$

$$\frac{d\pi_i(t)}{dt} = \lambda\pi_{i-1}(t) - \tag{44}$$

$$(\lambda + \mu)\pi_i(t) + \tag{45}$$

$$\mu\pi_{i+1}(t) \tag{46}$$

and for $i = 0$:

$$\frac{d\pi_0(t)}{dt} = -\lambda\pi_0(t) + \tag{47}$$

$$\mu\pi_1(t) \tag{48}$$

Stability condition

We focus on the steady-state analysis : so we suppose that under the stability condition, the number of customers when the time t is very large (t tends to infinity), has a limiting probability. We propose to solve the Chapman Kolmogorov equations in the steady state. So the derivatives in the Chapman Kolmogorov equations are null.

$$\frac{d\pi_i(t)}{dt} = 0$$

Steady state analysis

With the assumption that

$$\frac{d\pi_i(t)}{dt} = 0$$

For $i \geq 1$

$$\lambda\pi_{i-1} - \tag{49}$$
$$(\lambda + \mu)\pi_i + \tag{50}$$
$$\mu\pi_{i+1} = 0 \tag{51}$$

and for $i = 0$:

$$-\lambda\pi_0 + \tag{52}$$
$$\mu\pi_1 = 0 \tag{53}$$

Also we have the normalization condition :

$$\sum_i \pi_i = 1$$

Steady state analysis

Solving gives :

$$\pi_i = (\frac{\lambda}{\mu})^i \pi_0$$

Steady state probabilities

Stability condition :

$$\lambda < \mu$$

λ arrival rate μ service rate $\rho = \lambda/\mu$ is the load (or traffic intensity).

Steady state probabilities

$$\pi_i = \rho^i \pi_0$$

for $i \geq 1$ From the normalizing condition :

$$\sum_i \pi_i = \sum_i \rho^i \pi_0 = 1$$

$\pi_0 = \frac{1}{\sum_i \rho^i} = 1 - \rho$ if $\rho < 1$ Hence

$$\pi_i = \rho^i (1 - \rho)$$

for $i \geq 1$

Unstable queue

If $\lambda \geq \mu$ then $\rho \geq 1$ Arrival rate is greater than service rate and the queue is unstable. There is no steady state because the queue increases indefinitely.

Measurements

We define :

- mean number of customers N
- mean sojourn time R
- mean waiting time in the queue W
- mean throughput X
- mean utilization rate U

mean number of customers N

We take the expectation of $N(t)$

$$N = \sum_i i\pi_i = (1-\rho)\sum_i i\rho^i = (1-\rho)\rho\sum_i i\rho^{i-1}$$

$$N = (1-\rho)\rho\frac{1}{(1-\rho)^2}$$

$$N = \rho/(1-\rho)$$

mean throughput X

With the stability condition and capacity of the queue infinite :

$$X = \lambda$$

mean sojourn time R

Litlle's Law $X = N/R$

$$R = \frac{\rho}{(1-\rho)\lambda} = \frac{1}{\mu(1-\rho)}$$

$$R = 1/\mu + \frac{\rho}{\mu(1-\rho)} = S + W$$

with S mean service time and W mean waiting time in the queue

mean utilization rate U

at least one customer in the server so

$$U = 1 - \pi_0 = \rho$$

Example 1 : Demo with Pydev/Eclipse IDE and the Discrete Event Simulation library "Simpy" [13]

We consider a M/M/1 queue at a router with packets :
$mean\_interarrival\_time = 3$ and $mean\_service\_time = 1.7$

- what are the arrival rate and service rate ?
- Is there a mathematical formula for the following output metrics : mean waiting time in the queue, mean total waiting time, mean queue size ?
- If yes, give the formula and their numerical expressions.
- what about simulating such a queue ? what would be the results depending on the number of packets simulated ?

Example 1 : Demo with Pydev/Eclipse IDE and the Discrete Event Simulation library "Simpy" [13]

Eclipse : an IDE with different perspectives. Pydev perspective for Python language.
"SimPy is a discrete-event simulation library.
The behavior of active components (like vehicles, customers or messages) is modeled with processes.
All processes live in an environment. They interact with the environment and with each other via events.
"

Example 1 : Demo with Pydev/Eclipse IDE and the Discrete Event Simulation library "Simpy" [13]

"Processes are described by simple Python generators.
During their lifetime, they create events and yield them in order to wait for them to be triggered.
When a process yields an event, the process gets suspended.
SimPy resumes the process, when the event occurs (we say that the event is triggered).
Multiple processes can wait for the same event.
SimPy resumes them in the same order in which they yielded that event."

Example 1 : Demo with Pydev/Eclipse IDE and the Discrete Event Simulation library "Simpy" [13]

The first thing we need to do is to create an instance of Environment.
This instance is passed into our packet_generator and packet_consumer process functions.
Calling it creates a process generator that needs to be started and added to the environment via Environment.process().

```
# Setup and start the simulation
## The simulation environment.
env = simpy.Environment()
## The switch output port object based on the SimPy Sto
pipe = simpy.Store(env)
# Turns our generator functions into SimPy Processes
env.process(packet_generator(NUM_PACKETS, env, pipe))
env.process(packet_consumer(env, pipe))
```

Example 1 : Demo with Pydev/Eclipse IDE and the Discrete Event Simulation library "Simpy" [13]

Waiting for a Process.
As it happens, a SimPy Process can be used like an event (technically, a process actually is an event).
If you yield it, you are resumed once the process has finished, (looks like a function but with resuming). "Imagine a car-wash simulation where cars enter the car-wash and wait for the washing process to finish. "

Packet generator


```
def packet_generator(numPackets, env, out_pipe):
"""numPackets : int
        number of packets to send.
    env : simpy.Environment
        The simulation environment.
    out_pipe : simpy.Store
        the output queue model object.
"""
global queue_size
for i in xrange(numPackets):
    # wait for next transmission
    yield env.timeout(random.expovariate(1/ARRIVAL))
    #print "Sending packet {} at time {}".format(i, env.now)
    p = Packet(env.now, i)
    # Measuring queue statistics here is only valid for Pois
    queue_size += len(out_pipe.items)
    yield out_pipe.put(p)
```

Example 1 : Shared Resources

The resource's request() method generates an event that lets you wait until the resource becomes available again.
If you are resumed, you "own" the resource until you release it.
If you use the resource with the with statement as shown above, the resource is automatically being released.
If you call request() without with, you are responsible to call release() once you are done using the resource.
When you release a resource, the next waiting process is resumed and now "owns" one of the resource's slots.
The basic Resource sorts waiting processes in a FIFO (first in—first out) way.
A resource needs a reference to an Environment and a capacity when it is created.

Example 1 : Shared Resources


```
## The switch output port object based on the SimPy Store c
pipe = simpy.Store(env)

def packet_consumer(env, in_pipe):
    """
    Parameters
    ----------
    env : simpy.Environment
        the simulation environment.
    in_pipe : simpy.Store
        the FIFO model object where packets are kept.
    """
    global queue_wait, total_wait
    while True:
        # Get event for message pipe
        msg = yield in_pipe.get()
        queue_wait += env.now - msg.time
        yield env.timeout(random.expovariate(1/SERVICE))
        total_wait += env.now - msg.time
        #print "at time {} processed packet: {} ".format(env.n
```

Implementation Example 2 : A single server, M/M/1 queue

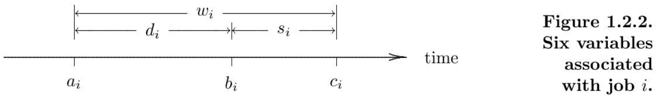


Figure – Notations taken from [14]

- arrival time of job i is a_i
- delay of job i in the queue is $d_i \geq 0$
- time that job i begins service is $b_i = a_i + d_i$
- service time of job i is s_i
- wait of job i in the service node (queue and service) is $w_i = d_i + s_i$
- time that job i completes service (the departure time) is $c_i = a_i + w_i$
- inter arrival time $r_i = a_i - a_{i-1}$

Implementation Example 2 : A single server, M/M/1 queue

We want to compute the delays d_i given the arrival times, the service times and the queue discipline. if the queue is not empty after job $i-1$ completes then :

$$
\begin{aligned}
c_{i-1} - a_i &= (a_{i-1} + d_{i-1} + s_{i-1}) - a_i && (54)\\
c_{i-1} - a_i &= d_{i-1} + s_{i-1} + (a_{i-1} - a_i) && (55)\\
c_{i-1} - a_i &= d_{i-1} + s_{i-1} - r_i && (56)\\
d_i &= d_{i-1} + s_{i-1} - r_i && (57)\\
& && (58)
\end{aligned}
$$

otherwise $d_i = 0$ Hence

$$d_i = \max\{d_{i-1} + s_{i-1} - r_i, 0\}$$

Implementation Example 2 : A single server, M/M/1 queue

Algorithm 1.2.1 If the arrival times $a_1, a_2, \ldots$ and service times $s_1, s_2, \ldots$ are known and if the server is initially idle, then this algorithm computes the delays $d_1, d_2, \ldots$ in a single-server FIFO service node with infinite capacity.

```
c0 = 0.0;                                  /* assumes that a0 = 0.0 */
i = 0;
while ( more jobs to process ) {
   i++;
   ai = GetArrival();
   if (ai < ci-1)
     di = ci-1 - ai;                        /* calculate delay for job i */
   else
     di = 0.0;                                     /* job i has no delay */
   si = GetService();
   ci = ai + di + si;            /* calculate departure time for job i */
}
n = i;
return d1, d2, ..., dn;
```

Figure – Algorithm taken from [14]

Implementation Example 2 : A single server, M/M/1 queue

job averaged statistics : average delay in the queue and average wait in the service node (in the queue and in service) :

$$\bar{w} = \frac{1}{n}\sum_{i=1}^{n} w_i = \frac{1}{n}\sum_{i=1}^{n}(d_i + s_i) = \bar{d} + \bar{s} \qquad (59)$$

average inter arrival time : $\bar{r} = \sum_{i=1}^{n} r_i = \sum_{i=1}^{n} a_n/n$

Implementation Example 2 : A single server, M/M/1 queue

- $l(t)$ the number of jobs in the service node a time t
- $q(t$ the number of jobs in the queue at time t
- $x(t)$ the number of jobs in service at time t

at any time t, $l(t) = q(t) + x(t)$ and the three functions are piecewise constant.

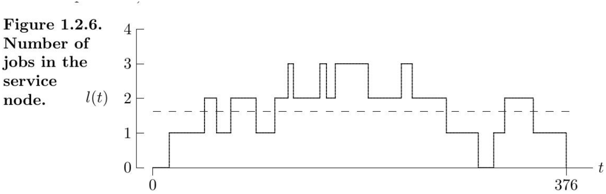


Figure – Output statistics taken from [14]

Implementation Example 2 : A single server, M/M/1 queue

During an interval $[0, \tau]$ the time averaged number of jobs in the node is :

$$\bar{l} = \frac{1}{\tau} \int_0^{\tau} l(t)dt$$

similary the time averaged number in the queue and in service are :

$$\bar{q} = \frac{1}{\tau} \int_0^{\tau} q(t)dt$$

$$\bar{x} = \frac{1}{\tau} \int_0^{\tau} x(t)dt$$

we have : $\bar{l} = \bar{q} + \bar{x}$

Implementation Example 2 : A single server, M/M/1 queue

Relation between job averaged and time averaged statistics : If the queue discipline is FIFO, the service node capacity is infinite, and the server is idle both initially and immediately after the departure of the nth job (at $t = c_n$) then

$$\int_0^{c_n} l(t)dt = \sum_{i=1}^{n} w_i$$

and

$$\int_0^{c_n} q(t)dt = \sum_{i=1}^{n} d_i$$

and

$$\int_0^{c_n} x(t)dt = \sum_{i=1}^{n} s_i$$

Implementation Example 2 : A single server, M/M/1 queue

proof : for each job i define an indicator function $\psi_i(t)$ that is 1 when the ith job is in the service node and 0 otherwise.
$\psi_i(t) = 1$ for $a_i < t < c_i$ and $\psi_i(t) = 0$ otherwise.
Then $l(t) = \sum_{i=1}^{n} \psi_i(t)$ for $0 < t < c_n$. And so

$$\int_0^{c_n} l(t)dt = \int_0^{c_n} \sum_{i=1}^{n} \psi_i(t)dt = \sum_{i=1}^{n} \int_0^{c_n} \psi_i(t)dt$$

$$= \sum_{i=1}^{n} (c_i - a_i) = \sum_{i=1}^{n} w_i$$

The two other equations can be derived in a similar way.

Traffic Modeling For Telecommunications Networks [9]

- Referent : communications systems
- Purpose : “As these new communications services evolve and the needs of users change, the enterprise must respond by modifying existing communications systems or by implementing entirely new ones.” ”Design and management decisions require predictions of network performance ;“
- Cost effectiveness : ” decisions based on poor predictions may adversely affect network customers ? perception of the new technology.“ ”a Monte Carlo simulation program can serve as a flexible testbed for conducting system experimentation without disturbing production networks or constructing software hardware prototypes.“

Traffic Modeling For Telecommunications Networks [9]

”For communications networks, developing a simulation program requires :

- Modeling random user demands for network resources.
- Characterizing network resources needed for processing those demands.
- Estimating system performance based on output data generated by the simulation.

“

A property of discrete event simulation

"Unlike analytical models that produce closed form answers, discrete event simulators often require large runs averaging over many events to converge to an answer. This may present practical difficulties in large simulations due to long simulation times. A variety of techniques can be applied to attempt to overcome these practical limitations. These techniques include the modeling of small sections of the system and approximating the overall performance of these sections by simpler "macro models" which are embedded in an overall system simulation. In this way, practical discrete event simulations can often be tailored to the accuracy requirements of the analysis. When run on a diagnostic basis, these simulations can identify the need for specific, more detailed "micro-level" simulations to validate the performance of critical portions of the system." [15].

Some definitions taken from [14]

"Briefly, a discrete-event simulation model is both stochastic and dynamic with the special discrete-event property that the system state variables change value at discrete times only (see Definition 1.1.1)." A simulation is said to be discrete in time if " the state of the system is a piecewise-constant function of time." [14].

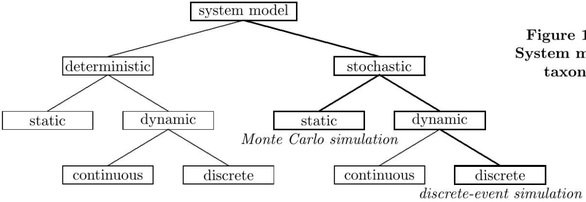


Figure 1.1.1. System model taxonomy.

Terminology

Material in this section is mainly taken from [14]. We will define :

- System state
- Events
- Simulation clock
- Events scheduling
- Event list (calendar)

State

Definition

The state of a system is a complete characterization of the system at an instance in time - a comprehensive "snapshot" in time. To the extent that the state of a system can be characterized by assigning values to variables, then state variables are what is used for this purpose.

Events

Definition

An event is an occurrence that may change the state of the system. By definition, the state of the system can only change at an event time. Each event has an associated event type.

Simulation clock

Definition

The variable that represents the current value of simulated time in a next-event simulation model is called the simulation clock.

Event scheduling

In a discrete-event simulation model it is necessary to use a time-advance mechanism to guarantee that events occur in the correct order - that is, to guarantee that the simulation clock never runs backward. The primary time-advance mechanism used in discrete-event simulation is known as next-event time advance ; this mechanism is typically used in conjunction with event scheduling.

Definition

If event scheduling is used with a next-event time-advance mechanism as the basis for developing a discrete-event simulation model, the result is called a next- event simulation model.

Next event simulation

To construct a next-event simulation model, three things must be done :

- construct a set of state variables that together provide a complete system description
- identify the system event types
- construct a collection of algorithms that define the state changes that will take place when each type of event occurs.

Event list

Definition

The data structure that represents the scheduled time of occurrence for the next possible event of each type is called the event list or calendar.

Next event simulation model

"Four steps compose the model :

- Initialize. The simulation clock is initialized (usually to zero) and, by looking ahead, the first time of occurrence of each possible event type is determined and scheduled, thereby initializing the event list.
- Process current event. The event list is scanned to determine the most imminent possible event, the simulation clock is then advanced to this event's scheduled time of occurrence, and the state of the system is updated to account for the occurrence of this event. This event is known as the "current" event.
- Schedule new events. New events (if any) that may be spawned by the current event are placed on the event list (typically in chronological order).
- Terminate. The process of advancing the simulation clock from one event time to the next continues until some terminal condition is satisfied. This terminal condition may be specified as a pseudo-event that only occurs once, at the end of the simulation, with the specification based on processing a fixed number of events, exceeding a fixed simulation clock time, or estimating an output measure to a prescribed precision.

"

Definitions
M/M/1 queue model
Simulation
Random number generation
In practical : Omnet++
Sumo

Material for this section taken from [14]

Desired properties of a random number generator (RNG)

A random number generator should be :

- *random* — able to produce output that passes all reasonable statistical tests of randomness;
- *controllable* — able to reproduce its output, if desired;
- *portable* — able to produce the same output on a wide variety of computer systems;
- *efficient* — fast, with minimal computer resource requirements;
- *documented* — theoretically analyzed and extensively tested.

Conceptual model of a random number generator

What we will do in this chapter is construct, from first principles, a good random number generator that will satisfy all the criteria listed previously. We begin with the following conceptual model.

- Choose a *large* positive integer m. This defines the set $\mathcal{X}_m = \{1, 2, \ldots, m-1\}$.
- Fill a (conceptual) urn with the elements of $\mathcal{X}_m$.
- Each time a random number u is needed, draw an integer x "at random" from the urn and let $u = x/m$.

Lehmer's random number generator

Definition 2.1.2 *Lehmer's algorithm* for random number generation is defined in terms of two fixed parameters

- *modulus* m, a fixed large *prime* integer
- *multiplier* a, a fixed integer in $\mathcal{X}_m$

and the subsequent generation of the integer sequence x_0, x_1, x_2, ... via the iterative equation

- $x_{i+1} = g(x_i) \qquad i = 0, 1, 2, \dots$

where the function $g(\cdot)$ is defined for all $x \in \mathcal{X}_m = \{1, 2, \dots, m-1\}$ as

- $g(x) = ax \bmod m$

and the *initial seed* x_0 is chosen from the set $\mathcal{X}_m$. The *modulus function* mod gives the remainder when the first argument (ax in this case) is divided by the second argument (the modulus m in this case). It is defined more carefully in Appendix B. A random number generator based on Lehmer's algorithm is called a *Lehmer generator*.

Full period considerations

From the definition of the mod function (see Appendix B), if $x_{i+1} = g(x_i)$ then there exists a non-negative integer $c_i = \lfloor ax_i/m \rfloor$ such that

$$x_{i+1} = g(x_i) = ax_i \bmod m = ax_i - mc_i \qquad i = 0, 1, 2, \ldots$$

Therefore (by induction)

$$\begin{aligned}
x_1 &= ax_0 - mc_0 \\
x_2 &= ax_1 - mc_1 = a^2x_0 - m(ac_0 + c_1) \\
x_3 &= ax_2 - mc_2 = a^3x_0 - m(a^2c_0 + ac_1 + c_2) \\
&\vdots \\
x_i &= ax_{i-1} - mc_{i-1} = a^ix_0 - m(a^{i-1}c_0 + a^{i-2}c_1 + \cdots + c_{i-1}).
\end{aligned}$$

Because $x_i \in \mathcal{X}_m$ we have $x_i = x_i \bmod m$. Moreover, $(a^ix_0 - mc) \bmod m = a^ix_0 \bmod m$ independent of the value of the integer $c = a^{i-1}c_0 + a^{i-2}c_1 + \cdots + c_{i-1}$. Therefore, we have proven the following theorem.

Theorem 2.1.1 If the sequence $x_0, x_1, x_2, \ldots$ is produced by a Lehmer generator with multiplier a and modulus m then

$$x_i = a^ix_0 \bmod m \qquad i = 0, 1, 2, \ldots$$

Full period considerations

Although it would be an eminently bad idea to compute x_i by first computing a^i, particularly if i is large, Theorem 2.1.1 has significant theoretical importance due, in part, to the fact that x_i can be written equivalently as

$$x_i = a^i x_0 \bmod m = [(a^i \bmod m)\,(x_0 \bmod m)] \bmod m = [(a^i \bmod m)\,x_0] \bmod m$$

for $i = 0, 1, 2, \ldots$ via Theorem B.2 in Appendix B. In particular, this is true for $i = m-1$. Therefore, because m is prime and $a \bmod m \neq 0$, from Fermat's little theorem (see Appendix B) it follows that $a^{m-1} \bmod m = 1$ and so $x_{m-1} = x_0$. This observation is the key to proving the following theorem. The details of the proof are left as an exercise.

Theorem 2.1.2 If $x_0 \in \mathcal{X}_m$ and the sequence $x_0, x_1, x_2, \ldots$ is produced by a Lehmer generator with multiplier a and (prime) modulus m then there is a positive integer p with $p \leq m-1$ such that $x_0, x_1, x_2, \ldots, x_{p-1}$ are all different and

$$x_{i+p} = x_i \qquad i = 0, 1, 2, \ldots$$

That is, the sequence $x_0, x_1, x_2, \ldots$ is *periodic* with *fundamental period* p. In addition, $(m-1) \bmod p = 0$.

Full period considerations

Theorem 2.1.3 If m is prime and $p_1, p_2, \ldots, p_r$ are the (unique) *prime factors* of $m-1$ then the number of full-period multipliers in $\mathcal{X}_m$ is

$$\frac{(p_1-1)(p_2-1)\ldots(p_r-1)}{p_1 p_2 \ldots p_r}\,(m-1).$$

Example 2.1.3 If $m = 13$ then $m-1 = 2^2 \cdot 3$. From the equation in Theorem 2.1.3 this prime modulus has $\frac{(2-1)(3-1)}{2\cdot 3}(13-1) = 4$ full-period multipliers: $a = 2$, 6, 7, and 11.

Example 2.1.4 The Lehmer generator used in this book has the (Mersenne, i.e., of the form $2^k - 1$, where k is a positive integer) prime modulus $m = 2^{31} - 1 = 2\,147\,483\,647$. Because the prime decomposition of $m-1$ is

$$m - 1 = 2^{31} - 2 = 2 \cdot 3^2 \cdot 7 \cdot 11 \cdot 31 \cdot 151 \cdot 331$$

from the equation in Theorem 2.1.3 the number of full-period multipliers is

$$\left(\frac{1 \cdot 2 \cdot 6 \cdot 10 \cdot 30 \cdot 150 \cdot 330}{2 \cdot 3 \cdot 7 \cdot 11 \cdot 31 \cdot 151 \cdot 331}\right)(2 \cdot 3^2 \cdot 7 \cdot 11 \cdot 31 \cdot 151 \cdot 331) = 534\,600\,000.$$

Therefore, for this prime modulus approximately 25% of the multipliers between 1 and $m-1$ are full-period multipliers.

Full period considerations

Theorem 2.1.4 If a is any full-period multiplier relative to the prime modulus m then each of the integers

$$a^i \bmod m \in \mathcal{X}_m \qquad i = 1, 2, 3, \ldots, m-1$$

is also a full-period multiplier relative to m if and only if the integer i has no prime factors in common with the prime factors of $m-1$, i.e., i and $m-1$ are relatively prime (see Appendix B).

Parameters choice

In 1988, Park and Miller suggested a Lehmer RNG with particular parameters $m = 2^{31} - 1 = 2147483647$(a Mersenne prime M31) and a = 75 = 16,807 (a primitive root modulo M31), now known as MINSTD.
Depending on the choice of the parameters, there will be different periods. For example, “Using a modulus m which is a power of two makes for a particularly convenient computer implementation, but comes at a cost : the period is at most m/4”

Inverse Transform Sampling


```
double exponential(cRNG *rng, double p)
{
    return -p *log(1.0 - rng->doubleRand());
}
```

p is the mean inter arrival time. doubleRand() function returns a random double on the [0,1] interval. why is this function written like that ?

Inverse Transform Sampling

https://en.wikipedia.org/wiki/Inverse_transform_sampling

"Let $F(x) = P(X \leq x)$ denote the distribution function of some random variable X. When F is continuous and strictly monotone increasing on the domain X, the random variable $U = F(X)$ with values in $[0,1]$ satisfies

$$P(U \leq u) = P(F(X) \leq u) = P(X \leq F^{-1}(u)) = F(F^{-1}(u)) = u$$

$\forall u \in [0,1]$

Thus, U is a uniform random variable over $[0,1]$, what we note $U \backsim U_{[0,1]}$. In other words, for all a, b with $0 \leq a \leq b \leq 1$, then $P(U \in [a,b]) = b - a$. Conversely, the distribution function of the random variable $Y = F^{-1}(U)$ is F when $U \backsim U_{[0,1]}$."

Inverse Transform Sampling

$$F(x) = P(X \leq x) = 1 - exp(-\lambda x)$$
$$F^{-1}(y) = -1/\lambda \times log(1 - y)$$

Demonstration

Vidéo de démonstration

Why learning Omnet++ ?

- "OMNeT++ is an object-oriented modular discrete event network simulation framework. It has a generic architecture, so it can be (and has been) used in various problem domains
- OMNeT++ provides infrastructure and tools for writing simulations
- OMNeT++ simulations can be run under various user interfaces
- OMNeT++ also supports parallel distributed simulation.
- OMNeT++ is used as a research and teaching tool, and there are many OMNeT++-based open-source simulation models and model frameworks released by various groups."

What is Omnet++ ? taken from [16]

"OMNeT++ is an object-oriented modular discrete event network simulation framework. It has a generic architecture, so it can be (and has been) used in various problem domains :

- modeling of wired and wireless communication networks
- protocol modeling
- modeling of queueing networks
- modeling of multiprocessors and other distributed hardware systems
- validating of hardware architectures
- evaluating performance aspects of complex software systems

in general, modeling and simulation of any system where the discrete event approach is suitable, and can be conveniently mapped into entities communicating by exchanging messages."

What is Omnet++ ?

"OMNeT++ itself is not a simulator of anything concrete, but rather provides infrastructure and tools for writing simulations. One of the fundamental ingredients of this infrastructure is a component architecture for simulation models. Models are assembled from reusable components termed "modules". Well-written modules are truly reusable, and can be combined in various ways like LEGO blocks."

What is Omnet++ ?

“Modules can be connected with each other via gates (other systems would call them ports), and combined to form compound modules. The depth of module nesting is not limited. Modules communicate through message passing, where messages may carry arbitrary data structures. Modules can pass messages along predefined paths via gates and connections, or directly to their destination ; the latter is useful for wireless simulations, for example. Modules may have parameters that can be used to customize module behavior and/or to parameterize the model's topology. Modules at the lowest level of the module hierarchy are called simple modules, and they encapsulate model behavior. Simple modules are programmed in C++, and make use of the simulation library. ”

What is Omnet++ ?

"OMNeT++ simulations can be run under various user interfaces. Graphical, animating user interfaces are highly useful for demonstration and debugging purposes, and command-line user interfaces are best for batch execution. The simulator as well as user interfaces and tools are highly portable. They are tested on the most common operating systems (Linux, Mac OS/X, Windows), and they can be compiled out of the box or after trivial modifications on most Unix-like operating systems."

What is Omnet++ ?

“OMNeT++ also supports parallel distributed simulation. OMNeT++ can use several mechanisms for communication between partitions of a parallel distributed simulation, for example MPI or named pipes. The parallel simulation algorithm can easily be extended, or new ones can be plugged in. Models do not need any special instrumentation to be run in parallel – it is just a matter of configuration. OMNeT++ can even be used for classroom presentation of parallel simulation algorithms, because simulations can be run in parallel even under the GUI that provides detailed feedback on what is going on. ”

Is OMNeT open-source ?

"OMNeT++ is used as a research and teaching tool, and there are many OMNeT++-based open-source simulation models and model frameworks released by various groups. OMNeT++ has its own web site at www.omnetpp.org. OMNeT++ is free for non-commercial activities, and it is released under the Academic Public License. This license provides non-commercial users of OMNeT++ with rights that are similar to the GNU General Public License. You are allowed to copy, modify and redistribute copies of OMNeT++ as long as you stay non-commercial. The license ensures that if you decide to use OMNeT++ for research or in teaching, you will always have OMNeT++ to back it, whatever happens to our company."

Modeling concepts

“An OMNeT++ model consists of modules that communicate with message passing. The active modules are termed simple modules ; they are written in C++, using the simulation class library. Simple modules can be grouped into compound modules and so forth ; the number of hierarchy levels is unlimited. The whole model, called network in OMNeT++, is itself a compound module. Messages can be sent either via connections that span modules or directly to other modules.”

Modeling concepts

In Fig. 2.1, boxes represent simple modules (gray background) and compound modules. Arrows connecting small boxes represent connections and gates.

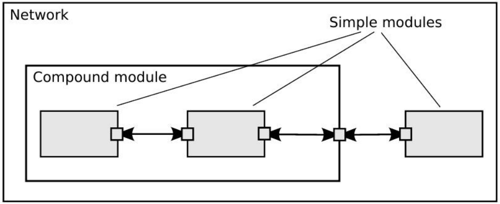


Figure 2.1: Simple and compound modules

Modeling concepts

"Modules communicate with messages that may contain arbitrary data, in addition to usual attributes such as a timestamp. Simple modules typically send messages via gates, but it is also possible to send them directly to their destination modules. Gates are the input and output interfaces of modules : messages are sent through output gates and arrive through input gates. An input gate and output gate can be linked by a connection. Connections are created within a single level of module hierarchy ; within a compound module, corresponding gates of two submodules, or a gate of one submodule and a gate of the compound module can be connected. Connections spanning hierarchy levels are not permitted, as they would hinder model reuse. Because of the hierarchical structure of the model, messages typically travel through a chain of connections, starting and arriving in simple modules."

Modeling concepts

"Parameters such as propagation delay, data rate and bit error rate, can be assigned to connections. One can also define connection types with specific properties (termed channels) and reuse them in several places. Modules can have parameters. Parameters are used mainly to pass configuration data to simple modules, and to help define model topology. Parameters can take string, numeric, or boolean values. Because parameters are represented as objects in the program, parameters – in addition to holding constants – may transparently act as sources of random numbers, with the actual distributions provided with the model configuration. They may interactively prompt the user for the value, and they might also hold expressions referencing other parameters. Compound modules may pass parameters or expressions of parameters to their submodules."

Modeling concepts

OMNeT++ provides efficient tools for the user to describe the structure of the actual system. Some of the main features are the following :

- hierarchically nested modules
- modules are instances of module types
- modules communicate with messages through channels
- flexible module parameters
- topology description language

Hierarchical modules

"An OMNeT++ model consists of hierarchically nested modules that communicate by passing messages to each other. OMNeT++ models are often referred to as networks. The top level module is the system module. The system module contains submodules that can also contain submodules themselves (Fig. 2.1). The depth of module nesting is unlimited, allowing the user to reflect the logical structure of the actual system in the model structure. Model structure is described in OMNeT++'s NED language. Modules that contain submodules are termed compound modules, as opposed to simple modules at the lowest level of the module hierarchy. Simple modules contain the algorithms of the model. The user implements the simple modules in C++, using the OMNeT++ simulation class library."

Messages, gates, links

"Modules communicate by exchanging messages. In an actual simulation, messages can rep- resent frames or packets in a computer network, jobs or customers in a queuing network or other types of mobile entities. Messages can contain arbitrarily complex data structures. Simple modules can send messages either directly to their destination or along a predefined path, through gates and connections. The "local simulation time" of a module advances when the module receives a message. The message can arrive from another module or from the same module (self-messages are used to implement timers). Gates are the input and output interfaces of modules ; messages are sent out through output gates and arrive through input gates. Each connection (also called link) is created within a single level of the module hierarchy."

Modeling of packet transmissions

"To facilitate the modeling of communication networks, connections can be used to model physical links. Connections support the following parameters : data rate, propagation delay, bit error rate and packet error rate, and may be disabled. These parameters and the underlying algorithms are encapsulated into channel objects. The user can parameterize the channel types provided by OMNeT++, and also create new ones."

Parameters

"Modules can have parameters. Parameters can be assigned in either the NED files or the configuration file **omnetpp.ini**. Parameters can be used to customize simple module behavior, and to parameterize the model topology. Parameters can take string, numeric or boolean values, or can contain XML data trees."

Topology Description Method

“The user defines the structure of the model in NED language descriptions (Network Descrip- tion).”

Programming the algorithms

"The simple modules of a model contain algorithms as C++ functions. The full flexibility and power of the programming language can be used, supported by the OMNeT++ simulation class library. The simulation programmer can choose between event-driven and process- style description, and freely use object-oriented concepts (inheritance, polymorphism etc) and design patterns to extend the functionality of the simulator. Simulation objects (messages, modules, queues etc.) are represented by C++ classes. They have been designed to work together efficiently, creating a powerful simulation programming framework."

Building and running simulations

"An OMNeT++ model consists of the following parts :

- NED language topology description(s) (.ned files) that describe the module structure with parameters, gates, etc. NED files can be written using any text editor, but the OMNeT++ IDE provides excellent support for two-way graphical and text editing.
- Message definitions (.msg files) that let one define message types and add data fields to them. OMNeT++ will translate message definitions into full-fledged C++ classes.
- Simple module sources. They are C++ files, with .h/.cc suffix.

"

Building and running simulations

The simulation system provides the following components :

- Simulation kernel. This contains the code that manages the simulation and the simulation class library. It is written in C++, compiled into a shared or static library.
- User interfaces. OMNeT++ user interfaces are used in simulation execution, to facilitate debugging, demonstration, or batch execution of simulations. They are written in C++, compiled into libraries.

Running the simulation and analyzing the results

"The simulation may be compiled as a standalone program executable, or as a shared library to be run using OMNeT++ opp_run utility. When the program is started, it first reads the NED files, then the configuration file usually called omnetpp.ini. The configuration file contains settings that control how the simulation is executed, values for model parameters, etc. The configuration file can also prescribe several simulation runs ; in the simplest case, they will be executed by the simulation program one after another. The output of the simulation is written into result files : output vector files, output scalar files, and possibly the user's own output files. OMNeT++ contains an Integrated Development Environment (IDE) that provides rich environment for analyzing these files. Output files are line-oriented text files which makes it possible to process them with a variety of tools and programming languages."

User interfaces

"The primary purpose of user interfaces is to make the internals of the model visible to the user, to control simulation execution, and possibly allow the user to intervene by changing variables/objects inside the model. This is very important in the development/debugging phase of the simulation project. Equally important, a hands-on experience allows the user to get a feel of the model's behavior. The graphical user interface can also be used to demonstrate a model's operation. The same simulation model can be executed with various user interfaces, with no change in the model files themselves."

Omnet++ : an example

network PacketQueueTutorialStep in project inet/queueing/tutorials/

```
network PacketQueueTutorialStep
{
    @display("bgb=600,200");
    submodules:
        producer: ActivePacketSource {
            @display("p=100,100");
        }
        queue: PacketQueue {
            @display("p=300,100");
        }
        collector: ActivePacketSink {
            @display("p=500,100");
        }
    connections allowunconnected:
        producer.out --> queue.in;
        queue.out --> collector.in;
}
```

Omnet++ : an example

parameters of the submodules of the network PacketQueueTutorialStep

```
[Config PacketQueue]
network = PacketQueueTutorialStep
sim-time-limit = 100000s

*.producer.packetLength = 1B
#*.producer.productionInterval = uniform(0s, 3s)
#*.collector.collectionInterval = uniform(0s, 1.7s)
```

Omnet++ : an example

module Active Packet Source

```
package inet.queueing.source;

import inet.queueing.base.PacketSourceBase;
import inet.queueing.contract.IActivePacketSource;

//
// This module is an active packet source which pushes packets into the connected
// packet consumer.
//
simple ActivePacketSource extends PacketSourceBase like IActivePacketSource
{
    parameters:
        volatile double productionInterval @unit(s); // elapsed time between subsequent packe
        @class(ActivePacketSource);
        @display("i=block/source");
        @signal[packetCreated](type=inet::Packet);
        @statistic[packetCreated](title="packets created"; record=count,sum(packetBytes),vect
    gates:
        output out @labels(push);
}
```

Omnet++ : an example

implementation of module Active Packet Source

```
#include "inet/common/ModuleAccess.h"
#include "inet/common/Simsignals.h"
#include "inet/queueing/source/ActivePacketSource.h"

namespace inet {
namespace queueing {

Define_Module(ActivePacketSource);

void ActivePacketSource::initialize(int stage)
{
    PacketSourceBase::initialize(stage);
    if (stage == INITSTAGE_LOCAL) {
        outputGate = gate("out");
        consumer = findConnectedModule<IPassivePacketSink>(outputGate);
        productionIntervalParameter = &par("productionInterval");
        productionTimer = new cMessage("ProductionTimer");
    }
    else if (stage == INITSTAGE_QUEUEING) {
        checkPushPacketSupport(outputGate);
        if (consumer == nullptr && !productionTimer->isScheduled())
            scheduleProductionTimer();
    }
}

void ActivePacketSource::handleMessage(cMessage *message)
{
    if (message == productionTimer) {
        if (consumer == nullptr || consumer->canPushSomePacket(outputGate->getPathEndGate())) {
            scheduleProductionTimer();
            producePacket();
        }
    }
    else
        throw cRuntimeError("Unknown message");
```

Omnet++ : an example

implementation of module Active Packet Source

```
void ActivePacketSource::producePacket()
{
    auto packet = createPacket();
    //packet->setArrivalTime(simTime()); //fix to measure queueing time
    EV_INFO << "Producing packet " << packet->getName() << "." << endl;
    pushOrSendPacket(packet, outputGate, consumer);
    updateDisplayString();
}

void ActivePacketSource::handleCanPushPacket(cGate *gate)
{
    Enter_Method("handleCanPushPacket");
    if (gate->getPathStartGate() == outputGate && !productionTimer->isScheduled()) {
        scheduleProductionTimer();
        producePacket();
    }
}

} // namespace queueing
} // namespace inet
```

Omnet++ : an example

module Active Packet Sink

```
package inet.queueing.sink;

import inet.queueing.base.PacketSinkBase;
import inet.queueing.contract.IActivePacketSink;

//
// This module is an active packet sink which pops packets from the connected
// packet provider. All popped packets are counted and deleted.
//
simple ActivePacketSink extends PacketSinkBase like IActivePacketSink
{
    parameters:
        volatile double collectionInterval @unit(s); // elapsed time between su
        @class(ActivePacketSink);
    gates:
        input in @labels(pop);
}
```

Omnet++ : an example

implementation of module Active Packet Sink

```
#include "inet/common/ModuleAccess.h"
#include "inet/common/Simsignals.h"
#include "inet/queueing/sink/ActivePacketSink.h"

namespace inet {
namespace queueing {

Define_Module(ActivePacketSink);

void ActivePacketSink::initialize(int stage)
{
    PacketSinkBase::initialize(stage);
    if (stage == INITSTAGE_LOCAL) {
        inputGate = gate("in");
        provider = findConnectedModule<IPassivePacketSource>(inputGate);
        collectionIntervalParameter = &par("collectionInterval");
        collectionTimer = new cMessage("CollectionTimer");
    }
    else if (stage == INITSTAGE_QUEUEING)
        checkPopPacketSupport(inputGate);
}

void ActivePacketSink::handleMessage(cMessage *message)
{
    if (message == collectionTimer) {
        if (provider->canPopSomePacket(inputGate->getPathStartGate())) {
            scheduleCollectionTimer();
            collectPacket();
        }
    }
    else
```

Omnet++ : an example

implementation of module Active Packet Sink

```
void ActivePacketSink::scheduleCollectionTimer()
{
    scheduleAt(simTime() + collectionIntervalParameter->doubleValue(), collectionTimer);
}

void ActivePacketSink::collectPacket()
{
    auto packet = provider->popPacket(inputGate->getPathStartGate());
    EV_INFO << "Collecting packet " << packet->getName() << "." << endl;
    numProcessedPackets++;
    processedTotalLength += packet->getDataLength();
    updateDisplayString();
    dropPacket(packet, OTHER_PACKET_DROP);
}

void ActivePacketSink::handleCanPopPacket(cGate *gate)
{
    Enter_Method("handleCanPopPacket");
    if (gate->getPathEndGate() == inputGate && !collectionTimer->isScheduled()) {
        scheduleCollectionTimer();
        collectPacket();
    }
}

} // namespace queueing
} // namespace inet
```

Omnet++ : an example

module Packet Queue

```
//
simple PacketQueue extends PacketQueueBase like IPacketQueue
{
    parameters:
        int packetCapacity = default(-1); // maximum number of packets in the queue, no li
        int dataCapacity @unit(b) = default(-1b); // maximum total length of packets in th
        string dropperClass = default(""); // determines which packets are dropped when th
        string comparatorClass = default(""); // determines the order of packets in the qu
        string bufferModule = default(""); // relative module path to the IPacketBuffer mo
        displayStringTextFormat = default("contains %p pk (%l) pushed %u\npopped %o remove
        @class(PacketQueue);
        @signal[packetPushed](type=inet::Packet);
        @signal[packetPopped](type=inet::Packet);
        @signal[packetRemoved](type=inet::Packet);
        @signal[packetDropped](type=inet::Packet);
        @statistic[packetPushed](title="packet pushed"; record=count,sum(packetBytes),vect
        @statistic[packetPopped](title="packet popped"; record=count,sum(packetBytes),vect
        @statistic[packetRemoved](title="packets removed"; record=count,sum(packetBytes),v
        @statistic[packetDropQueueOverflow](title="packet drops: queue overflow"; source=p
        @statistic[queueingTime](title="queueing times"; source=queueingTime(packetPopped)
        @statistic[queueLength](title="queue length"; source=count(packetPushed) - count(p
}
```

ign Source

Omnet++ : an example

implementation of module Packet Queue

```
#include "inet/common/ModuleAccess.h"
#include "inet/common/Simsignals.h"
#include "inet/queueing/function/PacketComparatorFunction.h"
#include "inet/queueing/function/PacketDropperFunction.h"
#include "inet/queueing/queue/PacketQueue.h"

namespace inet {
namespace queueing {

Define_Module(PacketQueue);

void PacketQueue::initialize(int stage)
{
    PacketQueueBase::initialize(stage);
    if (stage == INITSTAGE_LOCAL) {
        queue.setName("storage");
        inputGate = gate("in");
        producer = findConnectedModule<IActivePacketSource>(inputGate);
        outputGate = gate("out");
        collector = findConnectedModule<IActivePacketSink>(outputGate);
        packetCapacity = par("packetCapacity");
        dataCapacity = b(par("dataCapacity"));
        buffer = getModuleFromPar<IPacketBuffer>(par("bufferModule"), this, false);
        packetComparatorFunction = createComparatorFunction(par("comparatorClass"));
        if (packetComparatorFunction != nullptr)
            queue.setup(packetComparatorFunction);
        packetDropperFunction = createDropperFunction(par("dropperClass"));
    }
    else if (stage == INITSTAGE_QUEUEING) {
        checkPushPacketSupport(inputGate);
        checkPopPacketSupport(outputGate);
        if (producer != nullptr)
```

Omnet++ : an example

implementation of module Packet Queue

```
IPacketDropperFunction *PacketQueue::createDropperFunction(const char *dropperClass) const
{
    if (strlen(dropperClass) == 0)
        return nullptr;
    else
        return check_and_cast<IPacketDropperFunction *>(createOne(dropperClass));
}

IPacketComparatorFunction *PacketQueue::createComparatorFunction(const char *comparatorClass) const
{
    if (strlen(comparatorClass) == 0)
        return nullptr;
    else
        return check_and_cast<IPacketComparatorFunction *>(createOne(comparatorClass));
}

void PacketQueue::handleMessage(cMessage *message)
{
    auto packet = check_and_cast<Packet *>(message);
    pushPacket(packet, packet->getArrivalGate());
}

bool PacketQueue::isOverloaded() const
{
    return (packetCapacity != -1 && getNumPackets() > packetCapacity) ||
           (dataCapacity != b(-1) && getTotalLength() > dataCapacity);
}

int PacketQueue::getNumPackets() const
{
    return queue.getLength();
}
```

Omnet++ : an example

implementation of module Packet Queue

```
Packet *PacketQueue::getPacket(int index) const
{
    if (index < 0 || index >= queue.getLength())
        throw cRuntimeError("index %i out of range", index);
    return check_and_cast<Packet *>(queue.get(index));
}

void PacketQueue::pushPacket(Packet *packet, cGate *gate)
{
    Enter_Method("pushPacket");
    EV_INFO << "Pushing packet " << packet->getName() << " into the queue." << endl;
    queue.insert(packet);
    emit(packetPushedSignal, packet);
    if (buffer != nullptr)
        buffer->addPacket(packet);
    else if (isOverloaded()) {
        if (packetDropperFunction != nullptr) {
            while (!isEmpty() && isOverloaded()) {
                auto packet = packetDropperFunction->selectPacket(this);
                EV_INFO << "Dropping packet " << packet->getName() << " from the queue.\n";
                queue.remove(packet);
                dropPacket(packet, QUEUE_OVERFLOW);
            }
        }
        else
            throw cRuntimeError("Queue is overloaded but packet dropper function is not specified");
    }
    updateDisplayString();
    if (collector != nullptr && getNumPackets() != 0)
        collector->handleCanPopPacket(outputGate);
}
```

Omnet++ : an example

implementation of module Packet Queue

```
Packet *PacketQueue::popPacket(cGate *gate)
{
    Enter_Method("popPacket");
    auto packet = check_and_cast<Packet *>(queue.front());
    EV_INFO << "Popping packet " << packet->getName() << " from the queue." << endl;
    if (buffer != nullptr) {
        queue.remove(packet);
        buffer->removePacket(packet);
    }
    else
        queue.pop();
    emit(packetPoppedSignal, packet);
    updateDisplayString();
    animateSend(packet, outputGate);
    return packet;
}

void PacketQueue::removePacket(Packet *packet)
{
    Enter_Method("removePacket");
    EV_INFO << "Removing packet " << packet->getName() << " from the queue." << endl;
    if (buffer != nullptr) {
        queue.remove(packet);
        buffer->removePacket(packet);
    }
    else
        queue.remove(packet);
    emit(packetRemovedSignal, packet);
    updateDisplayString();
}
```

Omnet++ : an example

implementation of module Packet Queue

```
bool PacketQueue::canPushSomePacket(cGate *gate) const
{
    if (packetDropperFunction)
        return true;
    if (getMaxNumPackets() != -1 && getNumPackets() >= getMaxNumPackets())
        return false;
    if (getMaxTotalLength() != b(-1) && getTotalLength() >= getMaxTotalLength())
        return false;
    return true;
}

bool PacketQueue::canPushPacket(Packet *packet, cGate *gate) const
{
    if (packetDropperFunction)
        return true;
    if (getMaxNumPackets() != -1 && getNumPackets() >= getMaxNumPackets())
        return false;
    if (getMaxTotalLength() != b(-1) && getMaxTotalLength() - getTotalLength() < packet->getDataLength())
        return false;
    return true;
}

void PacketQueue::handlePacketRemoved(Packet *packet)
{
    Enter_Method("handlePacketRemoved");
    if (queue.contains(packet)) {
        EV_INFO << "Removing packet " << packet->getName() << " from the queue." << endl;
        queue.remove(packet);
        emit(packetRemovedSignal, packet);
        updateDisplayString();
    }
}

} // namespace queueing
} // namespace inet
```

Application Space

Simulation of Urban Mobility is

- a microscopic, space-continuous, and time-discrete traffic flow simulation.
- evolving to multi-modality
- suite of portable and modular binaries
- mainly developped by DLR, the national aeronautics and space research centre of the Federal Republic of Germany.

License

SUMO is distributed under GNU General Public License version 3.0 (GPLv3). It is a free software defined as a program who offers all of these freedoms :

- the freedom to use the software for any purpose
- the freedom to change the software to suit your needs
- the freedom to share the software with your friends and neighbors
- the freedom to share the changes you make

Distribution

GPL guarantees it will be free software and stay free software, no matter who changes or distributes the program (“viral” propagation) : copyleft = (!copyright).
When you distribute object code to users, GPL require you to provide all the source necessary to build the software.

Installation

- Compile from sources : successful builds on Linux, Windows, MacOs, Solaris, Cygwin.
- Install binaries : the way of doing it depends on the Operating System. For example on Debian and derivatives, this is enough :
  - apt install sumo

Versions

Version 0.24 was released on september 2, 2015
Link to : History of SUMO
Subversion is available to get a special old release (Use case : TranS)

SUMO is a collection of Binaries

They are modular, and portable. It includes tools to :

- build the network
- model the demand
- simulate the traffic
- monitor and control the simulation in real time
- analyze the results

Link to : Sumo 0.24 Tree

Build the network

It is possible to build it or import it. The tool is : netconvert
Demo :

- manual page
- importation from Open Street Map.
- build from edge, node, and additional files

Model the demand

- Demand is modeled with route files.
- A route file is a list of routes.
- A route is a list of edges the vehicle will go through.
- A route can be static or dynamic.

Tools for modeling the demand

- randomTrips.py : build random routes given a network
- od2trips : import o/d matrixes
- duarouter, jtrrouter, dfrouter : compute the routes (shortest path, from turning ratio, from detector values)

manual pages ?

Many car-following models

Link to : car Following Models

TraCI : Traffic Control Interface

- TCP client server architecture
- specification of communication protocol allows python, java, and web service interaction (TraaS) with SUMO.

TraCI : possibilities

- Value retrieving
- State changing

Link to : documentation

Simulation output

Many possibilities, for example :

- summary output
- floating car data output
- full output

References I

J. Rothenberg, L. E. Widman, K. A. Loparo, and N. R. Nielsen, “The nature of modeling,” *in Artificial Intelligence, Simulation and Modeling*, 1989.

R. E. Shannon, “Introduction to the art and science of simulation,” in *1998 Winter Simulation Conference. Proceedings (Cat. No.98CH36274)*, vol. 1, 1998, pp. 7–14 vol.1.

R. Frigg and S. Hartmann, “Models in science,” in *The Stanford Encyclopedia of Philosophy*, spring 2020 ed., E. N. Zalta, Ed. Metaphysics Research Lab, Stanford University, 2020.

References II

P.-A. Muller, F. Fondement, B. Baudry, and B. Combemale, "Modeling modeling modeling," *Software & Systems Modeling*, vol. 11, no. 3, pp. 347–359, Jul 2012. [Online]. Available : https://doi.org/10.1007/s10270-010-0172-x

A. W. Brown, "Model driven architecture : Principles and practice," *Software and Systems Modeling*, vol. 3, no. 4, pp. 314–327, 2004.

L. Kleinrock, "On the modeling and analysis of computer networks," *Proceedings of the IEEE*, vol. 81, no. 8, pp. 1179–1191, 1993.

A. A. B. Pritsker, J. O. Henriksen, P. A. Fishwick, and G. M. Clark, "Principles of modeling," Institute of Electrical and Electronics Engineers (IEEE), Tech. Rep., 1991.

References III

- A. A. B. Pritsker, "Compilation of definitions of simulation," *SIMULATION*, vol. 33, no. 2, pp. 61–63, 1979. [Online]. Available : https://doi.org/10.1177/003754977903300205
- V. S. Frost and B. Melamed, "Traffic modeling for telecommunications networks," *IEEE Communications Magazine*, vol. 32, no. 3, pp. 70–81, 1994.
- www.grotto-networking.com, *A Course on Network Optimization and Design*. [Online]. Available : https://www.grotto-networking.com/netDesignCourse.html
- R. G. Gallager, *Stochastic Processes : Theory for Applications*. Cambridge University Press, 2013.

References IV

Institut Mines Télécom, *Queuing Theory : from Markov Chains to Multi-Server Systems*. [Online]. Available : https://www.edx.org/course/queuing-theory-from-markov-chains-to-multi-server

Simpy. [Online]. Available : https://simpy.readthedocs.io/en/latest/

L. M. Leemis and S. K. Park, *Discrete-event simulation : A first course*. Pearson Prentice Hall Upper Saddle River, NJ, 2006.

References V

G. R. Larocque and S. J. Lipoff, "Application of discrete event simulation to network protocol modeling," in *Proceedings of ICUPC - 5th International Conference on Universal Personal Communications*, vol. 2, 1996, pp. 508–512 vol.2.

András Varga and Opensim Ltd, *Omnet++ simulation manual*, 2016.

Tufts university, *Introduction to Omnet++*. [Online]. Available : http://twl.ece.tufts.edu/twl/index.php

Prof. Dr. Mesut Gunes, FREIE UNIVERSITAT BERLIN, *Modeling and Performance Analysis with Simulation*. [Online]. Available : http://twl.ece.tufts.edu/twl/index.php

Thank you for your attention

Contact :

- Email : cyril.nguyen-van-phu@univ-eiffel.fr