NETWORKS AND TELECOMMUNICATIONS SERIES

Deterministic Network Calculus

From Theory to Practical Implementation

Anne Bouillard, Marc Boyer and Euriell Le Corronc

iSTE WILEY

Deterministic Network Calculus

Series Editor
Guy Pujolle

Deterministic Network Calculus

From Theory to Practical Implementation

Anne Bouillard
Marc Boyer
Euriell Le Corronc

ISTE
WILEY

First published 2018 in Great Britain and the United States by ISTE Ltd and John Wiley & Sons, Inc.

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

www.iste.co.uk

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

www.wiley.com

Library of Congress Control Number: 2018954538

British Library Cataloguing-in-Publication Data
A CIP record for this book is available from the British Library
ISBN 978-1-84821-852-9

Contents

Acknowledgments

The idea of writing a document that gathers existing results of network calculus stems from the French-founded project PEGASE (2010–2013). It took several years to write a complete book.

During these years, we were supported by many colleagues and friends, especially at ONERA and LINCS.

We would like to thank Éric Thierry for the fruitful discussion that we constantly had and who co-authored many contributions that we present in this book. We would also like to thank another co-author, Laurent Jouhet, from whom we took the idea of the section "Network calculus in four pages" (section I.3).

One of our objectives in writing this book was to clarify some aspects of network calculus. In particular, we would like to thank Guillaume Dufour, who is the initial author of the proofs of continuity insensitivity of section 5.3.3.3, as well as Thomas Calbas for his preliminary work on the EDF service policy of section 7.3.4 and Jörg Liebeherr, whose feedback on EDF was very valuable.

Finally, we would like to thank the colleagues who read the preliminary version of this book for their support and valuable comments. Cédric Mauclair reviewed a very preliminary version of this book, and Pierre Roux, Élie de Panafieu and Jean-Yves Le Boudec reviewed some quite final versions of the book.

Introduction

With the emergence of a multitude of network architectures, performance evaluation, originating from Erlang's work, has become an important research topic. The simplest model of a communication system is the *queue*, and by extension, these systems of queues have a high modeling power. A queue is composed of a *waiting queue* and a *server* that processes data in the waiting queue. If $X(t)$ is the number of data in the queue at time t, C is the number of data that can be processed by unit of time and $A(t)$ is the number of data entering the system between times t and $t+1$, then the first (and most important) formula that can be written is the *Lindley formula* [LIN 52]:

$$X(t+1) = \max(X(t) - C, 0) + A(t). \qquad \text{[I.1]}$$

Based on this formula, the queueing theory derives properties for $X(t)$ based on the knowledge of C and $A(t)$ given by some distributions and independence relations. Among the important results derived from this formula are the Pollaczek–Khinchine formula and the Little formula [LIT 61]. As far as stochastic models are concerned, queueing theory is still an active topic, and with the emergence of large and complex networks, new models have also been theory [BAC 02, LEB 07, GAS 12, DUF 10], stochastic geometry [BAC 09a, BAC 09b], random graphs [FRA 08, BOL 01, DRA 10] and so on.

Another direction of research is to make use of the "+" and the "$\max$" in equation [I.1]. Then, systems of queues can be analyzed using the (max,plus) (or tropical) algebra [BAC 92]. Compared to classical algebra, the addition is replaced by the maximum, and the multiplication by the addition. In this type of system, the addition models the time evolution and the maximum models the synchronization of events. An alternative and dual model for such systems uses the (min,plus) algebras, which is the base of the *network calculus* theory, the topic of this book.

Unlike classical queueing theory, it is based on the study of *envelopes* and bounding processes rather than studying their exact value. Introduced by the seminal work of Cruz [CRU 91a, CRU 91b], in which the traffic is characterized by (σ, ρ)-envelopes to compute maximum delays, the notion of envelope has been formalized and generalized to functions with values in the (min,plus) dioid [CRU 95]. The elements of a network, namely wires, switches and processors, have also been generalized from conservative link (the amount of data that can be served during each unit of time is constant) to more general envelopes in the same functional space. The aim of this theory is to compute deterministic upper bounds of performances (such as transmission delay or buffer usage) in networks.

In this model, data flows and systems are abstracted by functions in the dioid of the (min,plus) functions and performances can be derived by combining those functions through (min,plus) operators, such as the (min,plus) convolution, the (min,plus) deconvolution or the sub-additive closure. The main references on the topic are the textbooks [CHA 00] and [LEB 01], and since the pioneer works of Cruz in the 1990s, thousands of papers on network calculus have been published and at least six academic and four industrial tools based on network calculus have been developed.

The target application was originally communication networks, such as the Internet. Network calculus was successfully applied to study networks with differentiated services (DiffServ), as it enables us to compute a guaranteed rate for the best effort flows and integrated services (IntServ), which guarantees a bandwidth for each flow [PAR 93, PAR 94, FID 04]. Another success is the application of network calculus to define efficient load-balanced switches, the Birkhoff–Von Neumann switch, for example, which has a periodic scheme to connect any input to any output during a time proportional to the bandwidth requested for this connection [CHA 01, CHA 02a, CHA 02b]. A third example of application in this field is video-on-demand (VoD) [MCM 06, DUF 98, GAN 11].

However, in other fields of communication networks, dimensioning the capacities of the network with regard to the worst-case performance leads to over-provisioning. Indeed, the transmission times are often not critical: worst-case performance is not the right parameter to study (it happens very rarely), whereas the mean or the variability of the transmission delay might be the parameters to study, as the users of the network might be more sensitive to a *slower than usual* network. As a result, stochastic models are more relevant. For this aim, the stochastic counterpart of network calculus has been developed, the *stochastic network calculus* [CHA 94, JIA 08, FID 15]. The aim of this theory is to compute the violation probability of some flow of data to have a certain maximum delay. Several models have been defined, but they all mix the network calculus theory with the deviation theory.

Another class of applications where deterministic network calculus is relevant is real-time and critical systems. These systems have hard deadlines and require a deterministic analysis. The most famous success of network calculus is certainly its use to bound the delay and the buffer usage of the AFDX (Avionics Full Duplex) network of the Airbus A380 airplane [FRA 06, BOY 08, BOY 10c], and recent developments of network calculus have been obtained in this context. AFDX is an embedded network based on the Ethernet technology, and is where switches are connected using full-duplex links, i.e. there are two different wires between two switches, one for each direction, avoiding collisions. A realistic network is composed of a dozen switches and thousands of flows, called *virtual links*. As a result, the techniques developed to analyze such networks must be algorithmically efficient and compute accurate upper bounds on the transmission delays.

In the field of embedded networks, network calculus competes with other techniques. Among them, we can cite *model checking* [CLA 99]. Model checking is based on the exhaustive modeling of the states of the system with objects such as timed automata (or recently adapted to the context of network calculus, event-count automata [CHA 05]) and computes the exact bounds by analyzing them. As a result, it will give very accurate bounds, but at a prohibitive algorithmic cost. For example, in [PHA 07], a three-node network can be analyzed in 30 minutes. A second and classical technique is scheduling. In the context of the AFDX network, the *trajectorial approach* has been developed [MAR 06]. Given a sporadic flow (almost periodic with jitters) and a packet of this flow, the aim is to find a bound on the worst-case delay suffered by this packet given the interacting flows. The equation that gives this worst-case delay can then be solved using a fix-point equation. These techniques have been designed to be more accurate than the network calculus. Unfortunately, flaws have been found in this theory, invalidating the results of first investigations [KEM 13b, KEM 13a], and although these have been corrected [LI 14], no comparison with network calculus has been done since this correction.

In this book, we propose to present what are in our opinion the most important results obtained in the *deterministic* network calculus theory, since its first developments. Indeed, no general book on the topic has been published since the publication of two reference books in the early 2000s.

I.1. Organization of the book

This book is composed of three parts: the first presents the mathematical framework of the network calculus theory, the second focuses on the analysis of a network element in isolation and the third shows how to analyze a whole network.

These three parts are preceded by Chapter 1, which introduces the model and explains our assumptions in the rest of the book. We take the example of a single

server crossed by a single flow and show how the (min,plus) operators naturally appear in our model and how performances (delay and backlog) are computed.

The first part is about the mathematical framework, i.e. the dioid of (min,plus) functions:

– Chapter 2 defines this dioid and all the (min,plus) operators that will appear, namely the convolution, deconvolution and sub-additive closure. In this chapter, we use an algebraic approach whenever possible, to stress the importance of the dioid structure of the theory;

– Chapter 3 focuses on the classes of functions that are generally used for network calculus: functions are non-decreasing and important classes of functions are the convex and the sub-additive functions;

– Chapter 4 deals with the algorithmic aspects of the (min,plus) functions. Indeed, the implementation of the (min,plus) operators is important in order to build network calculus tools. We present general procedures to compute the (min,plus) operators and also some efficient algorithms when restricting to some classes of functions. We also present *containers* that enable us to efficiently perform the operations on approximate functions.

The second part defines the network calculus model of a server and is devoted to the local analysis of networks:

– Chapter 5 presents the foundations of the network calculus model, with the definitions of the arrival and service curves, that respectively model the data flows and the network elements. We show how performance bounds are computed from these curves;

– Chapter 6 extends the model to the case of a flow crossing a sequence of servers. The *pay burst only once* phenomenon, demonstrating the importance of studying a network in its globality, is explored. Several important use cases are presented, including the flow-window control;

– Chapter 7 presents the case of a network element crossed by several data flows. Here, the notion of *residual service curve* is introduced, allowing us to compute a service guarantee for every flow crossing the server. The residual service curve depends on the service policy of the server, and we study several of them;

– Chapter 8 extends Chapter 7 by considering packets instead of *fluid* data, showing the modeling power of network calculus;

– Chapter 9 ends the second part and offers a detailed comparison of the different notions of service curves that have been defined, including the *real-time calculus* theory.

The third part considers a whole network:

– Chapter 10 presents different solutions to compute end-to-end delay bounds in feed-forward networks. The *pay multiplexing only once* phenomenon is exhibited, which emphasizes the difficulty of computing accurate performance bounds in networks;

– Chapter 11 presents a completely different way to compute performance bounds in feed-forward networks, using linear programming. This method is less general, but when it applies, it allows us to compute tight performance bounds;

– Chapter 12 closes this third part by considering a network with cyclic dependencies. Sufficient conditions for the stability are computed, depending on the topology (with the fix-point method) or the service policy.

I.2. How to read this book

While writing this book, we had the following concerns in mind:

1) *self-satisfaction*: we tried to be as comprehensive as possible from the modeling with the (min,plus) algebra to the more elaborate results. We provide proofs for all of the results presented, although some have been simplified to keep them compact;

2) *rigor:* we paid attention to technical details such as function domains and continuity, while also justifying the modeling choices. Indeed, our personal experience is that most errors in formal methods come either from such details or from a modeling that does not capture the reality of the systems;

3) *audience diversity:* every reader has different expectations when opening a technical book. We tried to take into account this variety by presenting different aspects of network calculus: an engineer might be mainly interested in the modeling power and the applicability of the theory, but a developer might be more interested in its algorithmic aspects and in the tools that have been developed. Finally, some researchers might want to delve deeper into the theoretical aspects.

Figure I.1 represents the interdependence of the chapters.

A reader focusing on the application of network calculus could almost skip Part 1: they can refer to the definitions of the (min,plus) operators and classes of function in the summaries of Chapters 2 and 3 for the notations. Therefore, from Chapter 1 which motivates the model, they can jump to Chapters 5 to 8 and then to Chapter 10 and eventually to Chapter 12, depending on the type of networks they want to analyze. Finally, they might be interested in the list of existing tools in the Conclusion.

The reader interested in the implementation of network calculus will also be interested in the first part of the book, which deals with the operators and their properties, and especially in Chapter 4.

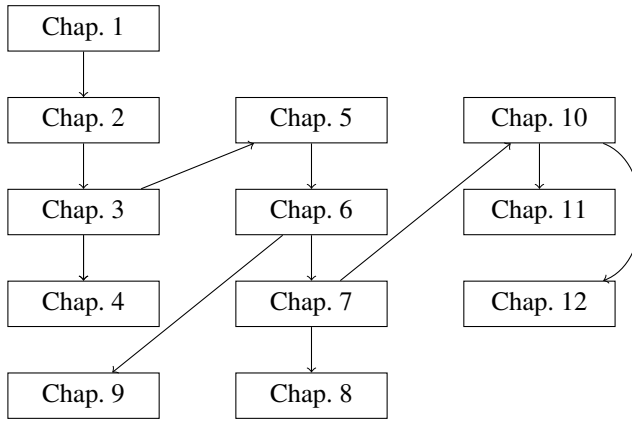

Figure I.1. *Dependence between chapters*

Some sections of this book are only of theoretical interest. This is particularly the case for Chapter 9, which compares the different types of service curves defined in the literature, and section 10.5, which proves the NP-hardness of computing tight performance bounds.

Finally, we shall use "WARNING.–" notes to emphasize counter-intuitive results.

I.3. Network calculus in four pages

In this section, we briefly present the model and the theory of network calculus. For readability, we do not mention the hypothesis required for the computations. The reader is referred to the rest of this book for more details.

Flow model: a flow of data crossing a given point of the network (from the input to the output of a server element) is described by a *cumulative function* A: $A(t)$ is the amount of data crossing that point up to time t.

Server model: a network element (or server), represented in Figure I.2, is a relation between two cumulative functions: A, the arrival cumulative function (at the input of the server) and D, the departure cumulative function (at the output of the server). For modeling purposes, this relation is usually not a function.

Performance bounds: From A and D, the arrival and departure cumulative functions, respectively, and provided that data exit in the same order as they arrive, the maximum delay $d(A, D)$ is the maximum horizontal distance between A and D, as illustrated in Figure I.3.

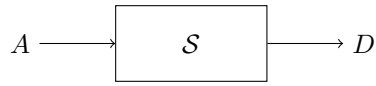

Figure I.2. *Server: a relation between arrival and departure cumulative functions*

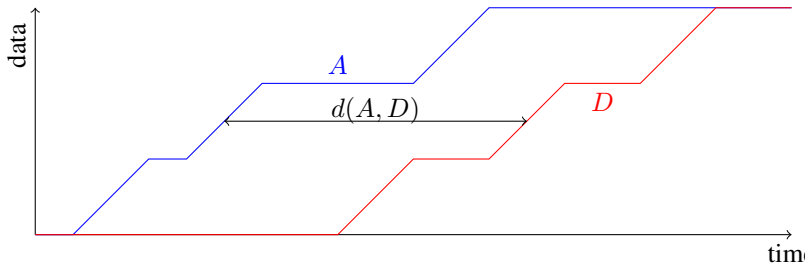

Figure I.3. *Delay from the cumulative functions. For a color version of this figure, see www.iste.co.uk/bouillard/calculus.zip*

An abstraction by curves: the exact behavior of a system is unknown at the design stage or too complex to be handled. The principle of network calculus is to abstract the flow and server by contracts, called *arrival curves* and *service curves*, respectively.

For the flows, the *arrival curve* bounds the amount of data during any interval of time: α is an arrival curve for A if

$$A(t+s) - A(t) \leqslant \alpha(s) \text{ for all } s \text{ and } t,$$

which translates into "the amount of data that arrived between time t and $t+s$ is less than $\alpha(s)$" and is illustrated in Figure I.4.

Symmetrically, a guarantee on the server is to bound the minimum amount of data that can be processed during any interval of time by the server. If $\beta(s)$ is the minimum amount of data that the server can process in any time interval of length s, then β is a *service curve*. Of course, as the server can be empty and thus serve no data, $D(t+s) - D(t)$ will not be lower-bounded by $\beta(s)$. However, we can show that D can be expressed as a function of A and β.

Algebraic relations: the (min,plus) convolution $*$ (defined in Chapter 2) enables us to relate the cumulative processes with the arrival and service curves:

$$A \leqslant A * \alpha \quad \text{and} \quad D \geqslant A * \beta. \qquad [I.2]$$

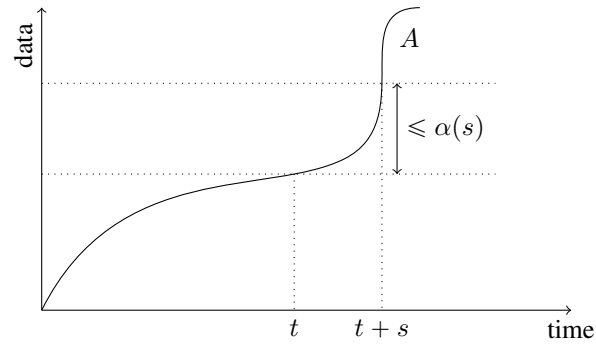

Figure I.4. *Arrival curve of a flow*

Performances from curves: from this modeling, it is now possible to compute bounds from the curves only. The maximum delay is bounded by the horizontal distance between α and β, and the maximum backlog (or buffer occupancy) by the vertical distance of the curves. Moreover, $\alpha \oslash \beta$ is an arrival curve for D, where $\oslash$ is the (min,plus) deconvolution which will be defined in Chapter 2.

From this modeling, we now illustrate how these basic elements can be used to analyze more complex networks.

Sequence of servers: suppose that the flow of data crosses several servers as in Figure I.5. Due to the nice algebraic properties of the (min,plus) convolution, we can write $C \geqslant B * \beta_2 \geqslant A * (\beta_1 * \beta_2)$, which means that the sequence of servers $\mathcal{S}_1$ and $\mathcal{S}_2$ can be modeled as a server with service curve $\beta_1 * \beta_2$.

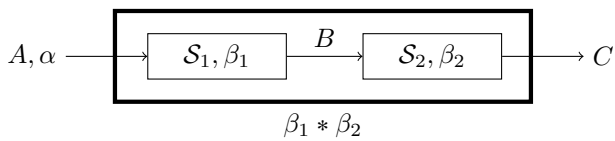

Figure I.5. *A single flow crossing two servers in tandem*

The end-to-end delay can then be computed by $d(\alpha, \beta_1 * \beta_2)$. Another solution would be to add the maximum delay of each server: $d(\alpha, \beta_1) + d(\alpha \oslash \beta_1, \beta_2)$.

The *pay burst only once* phenomenon, a key result of network calculus detailed in Chapter 6, is the observation that

$$d(\alpha, \beta_1 * \beta_2) \leqslant d(\alpha, \beta_1) + d(\alpha \oslash \beta_1, \beta_2).$$

Residual service curves: suppose now that the server is crossed by several flows. The service curve represents the global guarantee for the server that is shared among the flows. Our goal is to compute *residual service curves*, i.e. service guarantees for each flow. These service guarantees will of course depend on the service policy of the server (FIFO, priorities, processor sharing, etc.), and on the characteristics of the other flows.

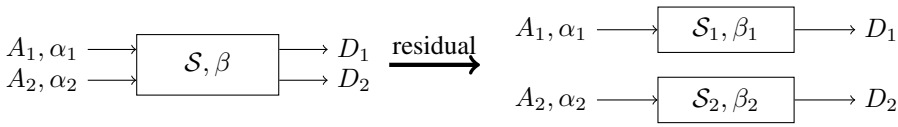

Figure I.6. *Residual service curves*

For example, when a server of capacity β is shared by two flows with respective arrival functions A_1, A_2 and arrival curves α_1, α_2, as in Figure I.6, the simplest result valid for any service policy is that a residual service offered to flow i is

$$\beta_i = [\beta - \alpha_j]^+_\uparrow ,$$

where $\{i, j\} = \{1, 2\}$ and $[x]^+_\uparrow$ will be defined in Chapter 3. This result seems quite simple and coherent with the intuition (each flow gets the minimal service minus the maximal interference), but technical assumptions have been omitted here and must be verified in Chapter 7.

Analysis of a network: let us now consider a more generic case as illustrated in Figure I.7.

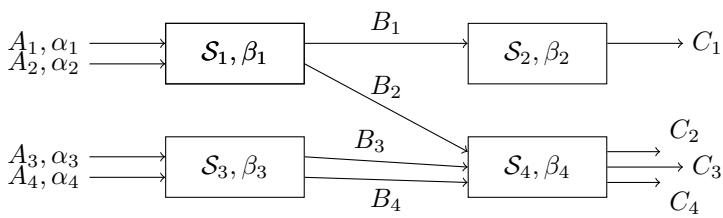

Figure I.7. *Topology with several flows crossing several servers*

To analyze such generic topologies, the most common approach, called *modular analysis*, consists of mixing results on the sequence of servers and residual service curves.

In Figure I.7, an end-to-end service curve for flow 1 (described by the cumulative functions A_1, B_1 and C_1) can be computed by

$$\tilde{\beta}_1 = [\beta_1 - \alpha_2]^+_\uparrow * \beta_2.$$

The case of flow 2 is a little more complicated as we also need to compute the residual service curve at server 4. This residual service curve can be computed by grouping flows 3 and 4: the end-to-end residual service curve for flow 2 is then:

$$\tilde{\beta}_2 = [\beta_1 - \alpha_1]^+_\uparrow * [\beta_4 - (\alpha_3 + \alpha_4) \oslash \beta_3]^+_\uparrow .$$

The case of flows 3 and 4 can be handled differently: the same process would be possible, but in many cases, it is possible to group servers 3 and 4 first: denote $\beta'_4 = \left[\beta_4 - \alpha_2 \oslash [\beta_1 - \alpha_1]^+_\uparrow\right]^+_\uparrow$ the residual service curve for flows 3 and 4 at server 4. Then, the residual service curve for flow 3 can be computed as

$$\tilde{\beta}_3 = \left[(\beta_3 * \beta'_4) - \alpha_3\right]^+_\uparrow ,$$

and similarly for flow 4. This illustrates the *pay multiplexing only once* phenomenon: the interference with flow 3 counted only once. The different modular approaches will be detailed in Chapter 10.

Another approach, explained in Chapter 11, consists of writing the equations on cumulative functions for the whole network and considering the arrival and service curves as constraints on the system. The computation of the delay then boils down to an optimization problem.

1

Basic Model: Single Server, Single Flow

Network calculus is a theory designed to compute upper bounds of flow delays and server backlogs in networks. Before presenting *how* to compute such bounds, we present in this chapter the modeling process so that the network calculus model is an abstraction of a *real* system. This includes modeling of the data that circulate in a network (flows) and modeling of the processing of data (servers). The modeling process also justifies the hypothesis we made in this book.

1.1. Modeling principles

Formal methods, as described in Figure 1.1, are used to compute properties in real systems (such as *there is no loss of message*): if a model $\mathcal{M}$ is built from a system Σ, then any property P of the system must be translated into a formal property Φ of the model. Such a property can, for example, be that there is no buffer overflow.

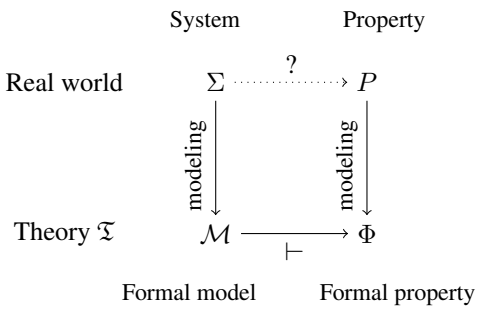

Figure 1.1. *Formal methods for guaranteeing properties on systems*

Deterministic Network Calculus: From Theory to Practical Implementation,
First Edition. Anne Bouillard, Marc Boyer and Euriell Le Corronc.

In our context, we want to ensure that if the model states that Φ is satisfied, then P is also satisfied. We then want a *conservative* modeling: it can never happen that a *good* property (i.e. the system satisfies all of the desired requirements) is satisfied in the model, but not in the real system. Indeed, the system's behavior would not be guaranteed.

Hence, we want to emphasize our modeling choices here and show how they fit systems.

1.2. Constant rate server

In this section, we present an example that will serve as an illustration in the whole chapter. Consider a server that transmits *exactly* R bits of data per unit time. Our aim is to find upper bounds on the transmission delay, assuming that data exit the server in their arrival order, or to bound the amount of data that can be present in the server.

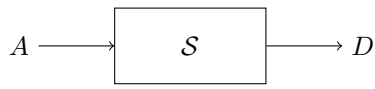

Figure 1.2. *Server: a relation between arrival and departure*

Let us first introduce some notations. For all $t \geqslant 0$, we denote by $A(t)$ the amount of data that arrive in the system up to time t, and by $D(t)$ the amount of data that departed the system up to time t. We assume that there is no loss and no creation of data in the system and that initially the system is empty: $A(0) = D(0) = 0$. The function A is called the *cumulative arrival function* (or *process*) of the system and D is called the *cumulative departure function* (or *process*) of the system. Let us now derive the relation between A and D.

Suppose that during the time interval $(u, t]$, the system is never empty (we also say that it is always backlogged), meaning that during this period of time, exactly $R \cdot (t - u)$ bits of data exit the system:

$$D(t) - D(u) = R \cdot (t - u). \qquad [1.1]$$

The assumption that the system is always backlogged is mandatory. Otherwise, we would have $D(t) - D(u) \leqslant R \cdot (t - u)$, which is not a guarantee for the service offered.

As $A(0) = D(0)$, the last instant s_0 before t at which the system is empty exists: $A(s_0) = D(s_0)$ and $s_0 = \sup\{s \leqslant t \mid A(s) = D(s)\}$. Using the latter formula, we obtain

$$D(t) = A(s_0) + R \cdot (t - s_0).$$

If $s < s_0$, then $D(t) \leqslant D(s) + R \cdot (t - s) \leqslant A(s) + R \cdot (t - s)$ as $D(s) \leqslant A(s)$.

If $s > s_0$, then $A(s) + R \cdot (t - s) \geqslant A(s) + D(t) - D(s) \geqslant D(t)$, and we finally obtain

$$D(t) = \inf_{0 \leqslant s \leqslant t} A(s) + R \cdot (t - s). \qquad [1.2]$$

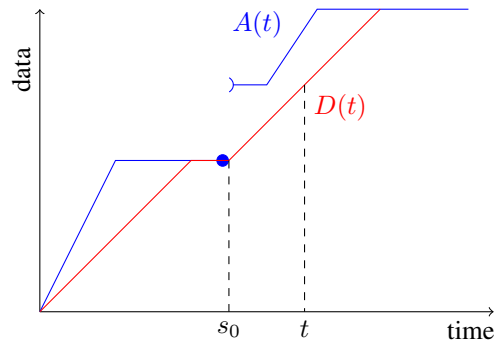

Figure 1.3. *Input/output relation for a constant-rate server. For a color version of this figure, see www.iste.co.uk/bouillard/calculus.zip*

This formula corresponds to the (min,plus) convolution of A and $\beta : t \mapsto R \cdot t$ that will be introduced in the next chapter. In short, we denote $D = A * \beta$.

Before explaining how performance guarantees can be computed, let us focus on the two key elements in the description of this system: the modeling of the flows and of the relation between the arrival and departure flows model (server).

1.3. Flow model

A data flow is usually a set of information, encoded into bits or bytes, grouped or fragmented into frames or packets, through the different network layer. Network calculus only focuses on performances, and parts of this information are forgotten. In particular, we focus only on the amount of information and not on its content, which in general does not affect the performance of the system. In particular, the size of the

packets is not modeled in the basic model, whereas it may have an impact depending on the scheduling policy. Chapter 8 will be devoted to such aspects.

In the basic model, a flow is represented by a cumulative function that models an *amount* of data.

DEFINITION 1.1 (Cumulative function).– *The* cumulative function *of a flow is a function A, defined on $\mathbb{R}^+$, that is non-decreasing, piecewise continuous and left-continuous and such that $A(0) = 0$. Denote by $\mathcal{C}$ the set of functions that satisfy these properties. The quantity $A(t)$ represents the amount of data sent by the flow in the interval $[0, t)$.*

An example of a cumulative function A is drawn in Figure 1.4. The slope between times u and u' may represent the fluid arrival of a packet of size 1, as well as the successive arrival of two packets of respective sizes $\frac{1}{3}$ and $\frac{2}{3}$. The interval $[u', v]$ not only could be the period between the arrival of two packets, but may also be some pause inside the arrival of one packet. The discontinuity at w can represent the instantaneous arrival of one or several packets: cumulative functions represent only an amount of data and give no information about packet boundaries or content.

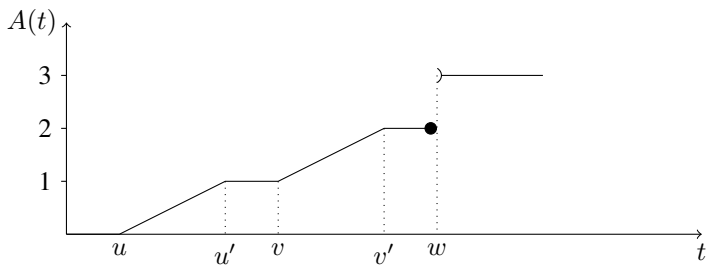

Figure 1.4. *Cumulative flow function*

After this short illustration, we more precisely discuss the following three points:

– the choice of the time domain;

– the use of a cumulative amount of data instead of throughput;

– the continuity of curves.

Time domain: the time domain is restricted to $\mathbb{R}^+$, excluding negative time values. We assume the existence of a *starting time* in the system: at time 0, all servers are empty and no flow has started sending data in the system.

The choice of a dense time domain might surprise some readers since a computer is basically a discrete-time system, scheduled by a clock. However, while considering a network, each element has its own clock, and different clocks may drift with regard to the others. As a result, the set of natural numbers for the time domain is excluded[1]. Still, we could choose $\mathbb{Q}^+$, but it is much more convenient to work with $\mathbb{R}^+$.

Cumulative amount of data: the choice of the cumulative amount of data, where the network community often represents a flow by its throughput, is mainly motivated by mathematical requirements. The amount of data sent by the flow from a time origin *up to the current time* t is always defined, whereas the *instantaneous throughput* is not defined at every time scale, especially because we assume that a positive amount of data may arrive instantaneously (as in Figure 1.4 at time w)

It is clear that a cumulative function is non-decreasing.

The left-continuity assumption: the point that deserves the longest discussion relies on the informal expression *up to time* t, and is related to the continuity of A.

There are several interpretations of this sentence: the quantity $A(t)$ represents the amount of data that arrived during $[0, t]$, $[0, t)$ or $(0, t]$.

The last case can be immediately discarded as $A(t)$ would never take into account the arrival of a packet at time 0 (even $A(0)$).

Initially, $[0, t]$ might seem more natural than $[0, t)$: with the right-open interval, a packet arriving at time t will not be taken into account in $A(t)$, but just after (at $A(t+) = \lim_{x \to t, x > t} A(t)$; see equation [A.1]); with the right-closed version, a packet arrival at time t is taken into account in $A(t)$.

The key point is that we are not only interested in the amount of data from 0 up to t, but also in any interval of time. Let s and t be two instants with $s \leqslant t$ and $A(s, t)$ be the amount of data arriving between times s and t. Intuitively, we want this amount of data to satisfy Chasles's relation:

$$\forall s \leqslant t \leqslant u,\ A(s, u) = A(s, t) + A(t, u),$$

and to be written as $A(s, t) = A(t) - A(s)$.

If $A(s, t)$ is the amount of data that arrived in the interval $[s, t]$, then we have $A(s, t) + A(t, u) = A(s, u) + A(t, t)$, where $A(t, t)$ might be positive. Therefore, we have to reject this model.

1 Note that while considering one single element, or a set of elements sharing a common clock, such as a system-on-chip, the assumption of a discrete time is a natural choice.

As has already been pointed out, if $A(s,t)$ is the amount of data that arrived in the interval $(s,t]$, then we have $A(s,t) + A(t,u) = A(s,u)$, but as mentioned above, the data that might arrive at time 0 will never be taken into account.

Therefore, finally, we consider $A(s,t)$ to be the amount of data that arrived in the interval $[s,t)$. We now have $A(t,t) = 0$ for all $t \in \mathbb{R}^+$, so Chasles's relation is satisfied, and we can set $A(t) = A(0,t)$:

$$A(0) = 0 \quad \text{and} \quad A(s,t) = A(0,t) - A(0,s) = A(t) - A(s).$$

This choice thus conforms to our requirements.

This choice implies two mathematical requirements on cumulative functions. First, it implies $A(0) = 0$ since the amount of data produced in the empty interval $[0,0)$ is null. Second, it implies left-continuous functions (the arrival of a single packet of size 1 at time 1 is represented by a function A such that $A(t) = 0$ for any $t \in [0,1]$ and $A(t) = 1$ for $t > 1$).

The last hypothesis, assuming piecewise continuity, is reasonable considering computer-based behavior: we assume that only a finite number of events can happen in a finite interval of time, so there is a finite number of discontinuities in any finite interval.

Nevertheless, after these presentations of the technical reasons why left-continuous cumulative curves are best suited to model cumulative amounts of data, we may wonder what would happen to the theory if right-continuous functions were chosen to model arrival curves. As already mentioned, this modeling seems more natural for many people since a packet arriving at time t is represented by the value of $A(t)$.

Section 5.3.3.3 will show that the choice of continuity for cumulative functions has no influence on the values of the delay and backlog, as they will be defined in section 1.5, and section 5.3.3.3 will provide more arguments that allow us to consider that continuity is a technical choice that can change the ways to get results, but not the results themselves.

1.4. Server model

In the example in section 1.2, the server was described as *transmitting exactly R bits of data per unit of time*, and we could derive a relation between the cumulative arrival function A and the cumulative departure function D. In network calculus, a server is a relation between some arrival and departure cumulative functions.