Technische Universität München
Lehrstuhl für Kommunikationsnetze
Prof. Dr.-Ing. Wolfgang Kellerer

Technical Report No. 201603

Network Calculus: A Comprehensive Guide

Amaury Van Bemten[1] and Wolfgang Kellerer[1]

[1]Chair of Communication Networks, Technische Universität München
Arcisstr. 21, 80333 München, Germany
{amaury.van-bemten|wolfgang.kellerer}@tum.de

October 8, 2016

Abstract

Network calculus is a mathematical framework allowing to analyze the worst-case performance of communication networks. As high performance is the goal of any communication network, we believe that the theory is a very useful tool for network researchers and engineers. However, because it relies on non-traditional algebras, namely the *min-plus* and *max-plus* algebras, researchers and engineers are usually reluctant to use network calculus or use it in a non-optimal or wrong way. Therefore, as an objective to make it more understandable and usable by the community, this document tries to present major results of network calculus in a comprehensive way. Proofs and detailed developments are intentionally omitted. We do not pretend to present any new result. Each statement is accompanied by a pointer to a proof or to a more detailed explanation for it. The goal of this document is to provide researchers and engineers with a comprehensive guide they can use as a reference to properly apply network calculus to their specific application.

Hoping for the best but expecting the worst

Contents

CHAPTER 1

INTRODUCTION

Network calculus is a system theory for communication networks. Based on the *min-plus* and *max-plus* algebras, network calculus gives a theoretical framework for analyzing performance guarantees in computer networks. Specifically, worst-case bounds on delay and buffer requirements in a network can be computed. Network calculus is said to be part of *exotic* or *tropical* algebras. These are a set of mathematical results giving insight into man-made systems (such as communication networks).

The results presented in this document focus on *deterministic* bounds computation, i.e. the document only deals with *deterministic network calculus*. Another branch of network calculus, namely *stochastic network calculus*, allows to compute worst-case performance bounds which follow probabilistic distributions. [Cha00], [JL08], [CBL05], [Fid06], [Jia06] and [Bac+92] are references covering this other branch of network calculus.

This document is mainly based on [BT12]. It tries to present results thereof in a more condensed and comprehensive way, clarifying, enhancing and sometimes correcting them with material from other publications. Whereas the focus of most network calculus references is on the demonstration and proof of the concepts they introduce, we focus here on their understandability and usability. Each statement however comes with a reference providing a proof or demonstration for it. Developments and modifications of results made by the authors of this guide are accompanied by a [$\star$] sign and, obviously, by a proof or explanation.

1.1 Network Calculus as a System Theory

As already mentioned, network calculus is the *system theory* that applies to computer networks. The main difference with the traditional system theory is that network calculus is based on min-plus algebra while traditional system theory is based on classical algebra. Roughly speaking, this means that the addition becomes the computation of the minimum and the multiplication becomes the addition.

Among the similarities between both theories is the usage of the *convolution* operator. In classical system theory, the convolution of an input signal by the impulse response of a system gives the output of the system. Also, the impulse response of the concatenation of a series of systems is given by the convolution of the impulse responses of all the systems. Similarly, in network calculus, the so-called min-plus convolution is used to compute the output of a traffic shaper or to merge nodes in series into one single node.

Nevertheless, both theories present some differences. A major one is the response of a linear system to the sum of two inputs. In classical system theory, the response to the sum of two inputs is the sum of the individual responses to each signal. In min-plus algebra, the addition corresponds to the multiplication in classical algebra and is therefore a non-linear operation. As a result, little is known on the aggregation of multiplexed flows. On the other hand, the computation of the minimum corresponds to the addition in classical algebra. Therefore, the operation is linear and the response to the minimum of two inputs is the minimum of the responses taken separately.

Another difference is how non-linear systems are handled. In classical system theory, non-linear systems are linearized around their operating point and the input signals are restricted around this operating point. In nework calculus, a non-linear system is replaced by a linear system that is a lower bound for the non-linear system. This is how worst-case performance measures can be computed.

1.2 Making Network Calculus Friendly

In order to provide worst-case performance bounds for flows in a network, network calculus requires a model of these flows and of the network. Once this modeling is done, the derivation of the bounds is an easy task. That is the bright side of network calculus. It gives easily *deterministic bounds*. Other tools, such as queueing theory, only allow to compute average values computed from probabilistic distributions. Deterministic bounds are for example necessary in real-time networking, where packets *have to* reach their destination before a pre-defined deadline. If one wants to guarantee no packet loss in its network, deterministic bounds (on the amount of data buffered at the different nodes) are also needed.

While network calculus provides useful results via simple derivations, it requires a complex first step: the modeling of the flows and of the network. While we will see in Chapter 4 that modeling flows is usually easy, the modeling of network nodes and the extension to multiples nodes, i.e. to a network, is more complex. First, though a node is usually modeled by a simple mathematical curve or function, ensuring that this curve is a correct model for this node is not trivial. Second, the extension to multiple nodes requires to apply some min-plus algebra, which is completely different from the classical algebra engineers are used to work with. Third, network calculus results are usually developed for bit-by-bit systems and therefore yield bounds which are not valid for packet-based systems. The step of converting the bounds to be valid for packet-based systems is often forgotten or neglected.

For these reasons, network calculus is seldom[1] used in the networking area and, when it is, it is difficult to ensure that it is done properly. As we believe it deserves much more attention, we try in this document to overcome these three obstacles and make network calculus easy to use and understand. In Chapter 2, we give a comprehensive, though classical, overview of the mathemical background. Chapters 3, 4 and 5 are then devoted to the description and explanation of how flows and network nodes are modeled. In Chapter 6, we then introduce how bounds can be derived from the models. As the extension from a single-hop to a multi-hop analysis requires min-plus algebra, we give examples of results in the commonly encountered cases, thereby

[1] Or not often enough compared to its usefulness.

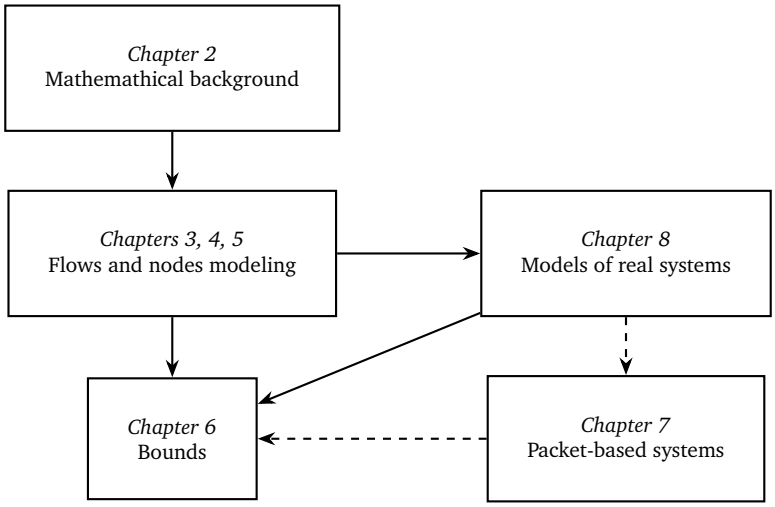


Figure 1.1: Organization of the Chapters of the report.

trying to allow the reader to circumvent the second issue mentioned here above. Then, while Chapter 7 explains how the transition from bit-by-bit systems to packet-based systems can be done, Chapter 8 presents models of commonly encountered networking nodes. The validity of these models for packet-based systems is highlighted. Thus, in combination with Chapter 7, Chapter 8 tackles both the first and third issues mentioned here above, allowing the reader to easily obtain packet-based models of the different nodes of its specific network setup. Then, after having modeled his flows, the reader can, using examples of Chapter 6, obtain the desired deterministic bounds. This structure is illustrated in Figure 1.1.

CHAPTER 2

MATHEMATICAL BACKGROUND

Before developing the results of network calculus theory, we first introduce the important underlying mathematical concepts. Network calculus is built on top of both min-plus and max-plus algebras. As it mainly uses *min-plus algebra*, we first focus on min-plus calculus. We then briefly introduce *max-plus algebra*.

[Bac+92] provides a detailed treatment of both min- and max-plus algebras. [Cha00] is an additional reference.

The reader can freely skip this Chapter and come back to it when corresponding mathematical results are used. Nevertheless, as the saying goes, *"cathedrals have not been built on sand"*. In other words, having a good understanding of the mathematical foundations is helpful to understand and use network calculus results properly.

2.1 Min-Plus Algebra

2.1.1 Introduction

In conventional algebra, one usually works with the algebraic structure $(\mathbb{R}, +, \times)$ (or $(\mathbb{Z}, +, \times)$), i.e. with the set of reals (or integers) endowed with the addition and multiplication operators. The particular properties of the two operators make $(\mathbb{R}, +, \times)$ a *commutative field* and $(\mathbb{Z}, +, \times)$ a *commutative ring* [BT12, pp. 103-104].

In min-plus algebra, the addition operator is replaced by the infimum (or the minimum if it exists)[1] operator ($\wedge$) and the multiplication operator is replaced by the addition operator. $+\infty$ is also included in the set of elements on which min-plus operators can be applied. This defines another algebraic structure, $(\mathbb{R} \cup \{+\infty\}, \wedge, +)$, which verifies the following properties [BT12, p. 105].

- **Closure of $\wedge$.**
  $\forall a, b \in \mathbb{R} \cup \{+\infty\}, \quad a \wedge b \in \mathbb{R} \cup \{+\infty\}$
- **Associativity of $\wedge$.**
  $\forall a, b, c \in \mathbb{R} \cup \{+\infty\}, \quad (a \wedge b) \wedge c = a \wedge (b \wedge c)$

[1] The *infimum* $\inf\{\mathcal{S}\}$ of a set $\mathcal{S}$ is defined as the greatest lower bound of $\mathcal{S}$. For example $\inf\{]0, 5]\} = 0$. By convention, $\inf\{\varnothing\} = +\infty$ [BT12, pp. 103-104].
The *minimum* $\min\{\mathcal{S}\}$ of a set $\mathcal{S}$ is the element of $\mathcal{S}$ which is smaller than all its other elements. It does not always exist. For example, $]0, 5]$ has no minimum [BT12, p. 103]. However, we have $\min\{[0, 5]\} = \inf\{[0, 5]\} = 0$.

- **Neutral element for $\wedge$.**
  $\exists e \in \mathbb{R} \cup \{+\infty\} : \quad \forall a \in \mathbb{R} \cup \{+\infty\}, \quad a \wedge e = a \qquad (e = +\infty)$
- **Idempotency of $\wedge$.**
  $\forall a \in \mathbb{R} \cup \{+\infty\}, \quad a \wedge a = a$
- **Commutativity of $\wedge$.**
  $\forall a, b \in \mathbb{R} \cup \{+\infty\}, \quad a \wedge b = b \wedge a$
- **Closure of $+$.**
  $\forall a, b \in \mathbb{R} \cup \{+\infty\}, \quad a + b \in \mathbb{R} \cup \{+\infty\}$
- **Associativity of $+$.**
  $\forall a, b, c \in \mathbb{R} \cup \{+\infty\}, \quad (a + b) + c = a + (b + c)$
- **Neutral element for $+$.**
  $\exists u \in \mathbb{R} \cup \{+\infty\} : \quad \forall a \in \mathbb{R} \cup \{+\infty\}, \quad a + u = a = u + a \qquad (u = 0)$
- **Commutativity of $+$.**
  $\forall a, b \in \mathbb{R} \cup \{+\infty\}, \quad a + b = b + a$
- **The neutral element for $\wedge$ is absorbing for $+$.**
  $\forall a \in \mathbb{R} \cup \{+\infty\}, \quad a + e = e = e + a$
- **Distributivity of $+$ with respect to $\wedge$.**
  $\forall a, b, c \in \mathbb{R} \cup \{+\infty\}, \quad (a \wedge b) + c = (a + c) \wedge (b + c) = c + (a \wedge b)$

These axioms define a *commutative dioid*[2] [BT12, p. 105]. It is not a (commutative) ring because the $\wedge$ operator is idempotent rather than cancelable [BT12, p. 105]. For example, $(\mathbb{Z}, +, \times)$ is a (commutative) ring because the $+$ operator is cancelable. It is not a (commutative) field because it does not contain a multiplicative inverse for every non-zero element. $(\mathbb{R}, +, \times)$ is therefore a (commutative) field.

2.1.2 Wide-Sense Increasing Functions

A function or sequence[3] f is said *wide-sense increasing* if and only if [BT12, p. 105]

$$\boxed{\forall s \geq t, \quad f(s) \geq f(t)}. \qquad \text{(Wide-Sense Increasing Function)}$$

We adopt the following notations [BT12, p. 105].

- $\mathcal{G}$: set of non-negative wide-sense increasing functions or sequences.
- $\mathcal{F}$: set of non-negative wide-sense increasing functions or sequences such that $f(t) = 0$ if $t < 0$.

2.1.3 Pseudo-Inverse of Wide-Sense Increasing Functions

It is well known that strictly increasing (or decreasing) functions are invertible [AE10, p. 164]. However, wide-sense increasing functions are more general functions

[2] It would have been a simple *dioid* if the $+$ operator was not commutative.
[3] $f(t)$ is called a *function* when its parameter t is continuous and a *sequence* when t is discrete.

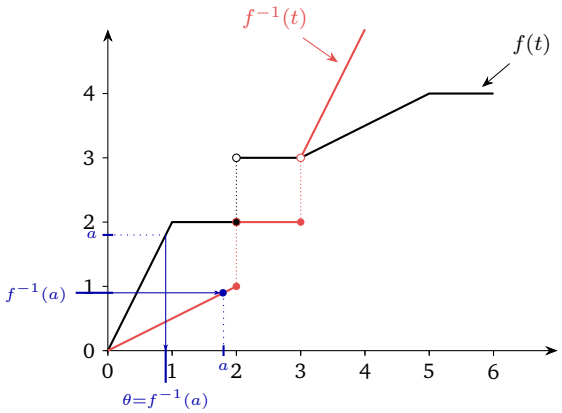


Figure 2.1: Derivation of the pseudo-inverse of a function $f \in \mathcal{F}$.

and are not always invertible. Therefore, the *pseudo-inverse* f^{-1} of a function $f \in \mathcal{F}$ is defined as [BJT09, p. 5][4]

$$\boxed{f^{-1}(x) = \inf_{t \geq 0}\{t : f(t) \geq x\}}. \qquad \text{(Pseudo-Inverse of } f \in \mathcal{F}\text{)}$$

The concept is illustrated in Figure 2.1. Let us see how the value $f^{-1}(a)$ for a given a can be computed. First, we draw, from left to right, a horizontal line at ordinate value a. When we reach an abscissa θ for which $f(\theta) \geq a$, we stop. θ is then the value of the pseudo-inverse of f in a, i.e. $\theta = f^{-1}(a)$.

The Figure shows that $f^{-1}(x)$ can actually be obtained by connecting all discontinuity points of $f(x)$ and then mirroring the result around the axis $y = x$. However, doing so, since f is not strictly increasing, some abscissas will have several corresponding ordinates. For these border cases, the pseudo-inverse value must then be explicitly computed as described above.

The pseudo-inverse is a closed operation in $\mathcal{F}$, $f^{-1}(0) = 0$ and we have [BT12, pp. 108-109]

$$y \leq f(x) \quad \Rightarrow \quad f^{-1}(y) \leq x \tag{2.1}$$

$$x > f^{-1}(y) \quad \Rightarrow \quad f(x) \geq y \tag{2.2}$$

It can also be shown that

$$f^{-1}(x) = \sup_{t \geq 0}\{t : f(t) < x\} \tag{2.3}$$

is an equivalent definition [BT12, pp. 108-109].

2.1.4 Concave, Convex and Star-Shaped Functions

Concave, convex and star-shaped functions are of particular interest in network calculus. They are defined as follows.

[4]Note that the definition in [BT12, p. 108] does not include the restriction that $t \geq 0$. However, this is crucial for the properties mentioned later to be true [⋆]. For example, with the definition $f^{-1}(x) = \inf\{t : f(t) \geq x\}$ of [BT12, p. 108], we have $f^{-1}(0) = -\infty$ for $f \in \mathcal{F}$. Therefore, $f^{-1} \notin \mathcal{F}$ and the pseudo-inverse operator is not closed in $\mathcal{F}$.

A function $f : \mathbb{R} \to \mathbb{R}$ is *convex* if and only if [BT12, p. 109]

$$\boxed{\forall x, y \in \mathbb{R}, u \in [0,1], \quad f(ux + (1-u)y) \leq uf(x) + (1-u)f(y)}.$$
(Convex Function)

A function $f : \mathbb{R} \to \mathbb{R}$ is *concave* if and only if $-f$ is convex or, alternatively, if and only if [BT12, p. 109]

$$\boxed{\forall x, y \in \mathbb{R}, u \in [0,1], \quad f(ux + (1-u)y) \geq uf(x) + (1-u)f(y)}.$$
(Concave Function)

A function $f : [0, +\infty[\to \mathbb{R}$ is *star-shaped* if and only if [BO62, pp. 1203-1204]-[MOA11, p. 650]

$$\boxed{\forall x \geq 0, \alpha \in [0,1], \quad f(\alpha x) \leq \alpha f(x)}, \qquad \text{(Star-Shaped Function)}$$

or, alternatively, if $f(0) \leq 0$ and $\forall x > 0, \ \ f(x)/x$ is wide-sense increasing [MOA11, p. 650]-[BO62, p. 1205].

These mathematical definitions are not very intuitive. Actually, these are based on the definition of convex and star-shaped domains. A domain $\mathcal{S} \subseteq \mathbb{R}^n$ is *convex* if and only if [BT12, p. 109]

$$\boxed{\forall x, y \in \mathcal{S}, u \in [0,1], \quad ux + (1-u)y \in \mathcal{S}}. \qquad \text{(Convex Set)}$$

Intuitively, a set $\mathcal{S}$ is convex if the line segment connecting any two points of $\mathcal{S}$ is entirely in $\mathcal{S}$.

A domain $\mathcal{S} \subseteq \mathbb{R}^n$ is *star-shaped* (or is a *star-domain*) if and only if [ST83, pp. 141-142]

$$\boxed{\exists x_0 \in \mathcal{S} : \quad \forall x \in \mathcal{S}, u \in [0,1], \quad ux_0 + (1-u)x \in \mathcal{S}}. \qquad \text{(Star-Shaped Set)}$$

Intuitively, a set $\mathcal{S}$ is star-shaped if there is a point x_0 in $\mathcal{S}$ such that the line segment connecting x_0 to any other point in $\mathcal{S}$ is entirely in $\mathcal{S}$. The set is then also said *star-shaped with respect to* x_0.

Examples of such domains in $\mathbb{R}^2$ are shown in Figure 2.2. The set $\mathcal{C}$ and $\mathcal{S}$ are respectively convex and star-shaped. The set $\mathcal{D}$ and $\mathcal{T}$ are respectively not convex and not star-shaped because of the existence of the dashed line segment violating the definitions here above. Note that $\mathcal{D}$ and $\mathcal{C}$ are also star-shaped.

From this, we can now give more intuitive definitions of convex, concave and star-shaped functions.

A function $f : \mathbb{R} \to \mathbb{R}$ is *convex* if and only if its epigraph[5] is a convex set [MOA11, p. 646].

[5]The epigraph of a function is the set of points lying above the graph of the function.

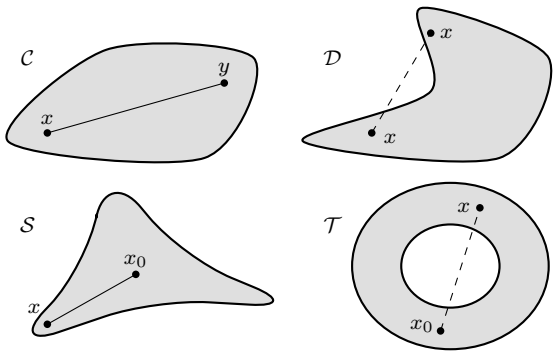


Figure 2.2: $\mathcal{C} \subset \mathbb{R}^2$ is a convex set because the line connecting any two points of $\mathcal{C}$ is entirely in $\mathcal{C}$. In contrast, $\mathcal{D} \subset \mathbb{R}^2$ is not a convex set. $\mathcal{S} \subset \mathbb{R}^2$ is a star-shaped set because there exist a point $x_0 \in \mathcal{S}$ such that the line segment connecting x_0 to any other point in $\mathcal{S}$ is entirely in $\mathcal{S}$. In contrast, $\mathcal{T} \subset \mathbb{R}^2$ is not a star-shaped set.

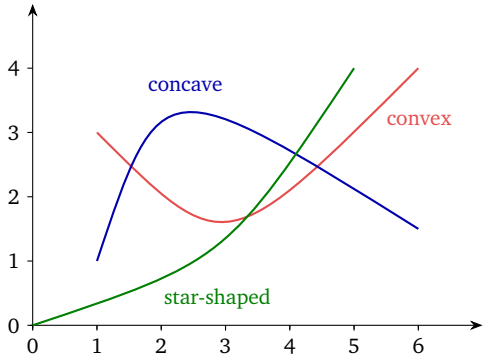


Figure 2.3: Example of concave, convex and star-shaped functions.

A function $f : \mathbb{R} \to \mathbb{R}$ is *concave* if and only if its opposite $-f$ is convex or, alternatively, if the set of points lying below its graph is a convex set.

A function $f : [0, +\infty[\to \mathbb{R}$ is *star-shaped* if and only if the set of points lying above its graph is star-shaped with respect to the origin [BO62, p. 1203]. Note that the star-shaped property of a function is only defined for functions whose domain is $[0, +\infty[$. We will however use it for functions defined on the whole real line by considering only the values of the function in $[0, +\infty[$.

Figure 2.3 shows examples of such functions. These functions enjoy some interesting properties.

- The maximum (resp. minimum) of any number of convex (resp. concave) functions is a convex (resp. concave) function [BT12, p. 109], that is

$$f_1(t), \ldots, f_n(t) \text{ are convex } \Rightarrow g(t) = \max_{1 \leq i \leq n} \{f_i(t)\} \text{ is convex,} \tag{2.4}$$

$$f_1(t), \ldots, f_n(t) \text{ are concave } \Rightarrow g(t) = \min_{1 \leq i \leq n} \{f_i(t)\} \text{ is concave.} \tag{2.5}$$

- If $f(0) \leq 0$ and f is convex, then f is star-shaped [MOA11, p. 650]-[BO62, p. 1207]. Similarly, as can be seen in Figure 2.2, a convex set is star-shaped.

- The maximum of two star-shaped functions is star-shaped [BT12, p. 110].

Note that in [BT12], the authors consider that a function $f \in \mathcal{F}$ is star-shaped if the set of points lying *below* its graph is star-shaped with respect to the origin. As we did not find any naming for such functions in the literature, we will call them *lower-star-shaped* functions [$\star$]. We think they do this because, in network calculus, lower-star-shaped functions are more often used than star-shaped functions. Obviously, a function f is lower-star-shaped if its opposite $-f$ is star-shaped and lower-star-shaped functions therefore enjoy properties similar to star-shaped functions:

- if $f(0) \geq 0$ and f is concave, then f is lower-star-shaped;
- the minimum of two lower-star-shaped functions is lower-star-shaped.

2.1.5 Min-Plus Convolution

In classical system theory using classical algebra, the *convolution* of functions $f(t)$ and $g(t)$ (that are zero for $t < 0$) is defined as

$$(f * g)(t) = \int_0^t f(t-s)g(s)\, ds. \qquad \text{(Convolution)}$$

Transforming the addition into infimum and the multiplication into addition, we obtain the definition of the *min-plus convolution* for $f, g \in \mathcal{F}$.

$$\boxed{(f \otimes g)(t) = \inf_{0 \leq s \leq t} \{f(t-s) + g(s)\}} \qquad \text{(Min-Plus Convolution)}$$

(If $t < 0$, $(f \otimes g)(t) = 0$.)

The min-plus convolution enjoys the following properties in $\mathcal{F}$ [BT12, pp. 111-115].

- **Closure.**
  $\forall f, g \in \mathcal{F}, \quad f \otimes g \in \mathcal{F}$
- **Associativity.**
  $\forall f, g, h \in \mathcal{F}, \quad (f \otimes g) \otimes h = f \otimes (g \otimes h)$
- **Neutral element.**
  $\exists \delta_0 \in \mathcal{F} : \quad \forall f \in \mathcal{F}, \quad f \otimes \delta_0 = f \qquad (\delta_0 = +\infty \text{ if } t > 0, 0 \text{ otherwise})$
- **Commutativity.**
  $\forall f, g \in \mathcal{F}, \quad f \otimes g = g \otimes f$
- **Distributivity with respect to $\wedge$.**
  $\forall f, g, h \in \mathcal{F}, \quad (f \wedge g) \otimes h = (f \otimes h) \wedge (g \otimes h)$
- **Addition of a constant.**
  $\forall f, g \in \mathcal{F}, K \in \mathbb{R}^+, \quad (f + K) \otimes g = (f \otimes g) + K$
- **Isotonicity.**
  $\forall f, g, f', h' \in \mathcal{F}, \quad f \leq g, f' \leq g' \;\Rightarrow\; f \otimes f' \leq g \otimes g'$
- $\forall f, g \in \mathcal{F}, \quad f(0) = g(0) = 0 \;\Rightarrow\; f \otimes g \leq f \wedge g$
- $\forall f, g \in \mathcal{F}, \quad f(0) = g(0) = 0$ and f, g lower-star-shaped $\;\Rightarrow\; f \otimes g = f \wedge g$

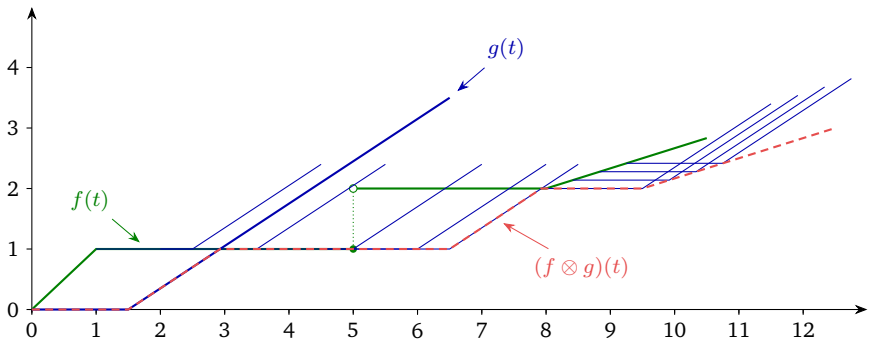


Figure 2.4: The min-plus convolution of two functions passing through the origin can be obtained by placing one of the two functions at each point of the other function and taking the minimum of all the resulting functions.

- $\forall f, g \in \mathcal{F}, \quad f, g$ convex $\Rightarrow f \otimes g$ convex. In particular, if f and g are convex and piecewise linear, their convolution is obtained by putting end-to-end the different linear pieces of f and g, sorted by increasing slopes.

Graphically, the min-plus convolution of two functions $f(t) \in \mathcal{F}$ and $g(t) \in \mathcal{F}$ can be computed as shown in Figure 2.4. Let us first consider the case where $f(0) = g(0) = 0$. In this case, the min-plus convolution is obtained by placing g at each point of f and keeping the minimum of all these functions and of f. If the functions are not zero at the origin, the *Addition of a constant* property here above shows that we can apply the construction explained above on $f(t) - f(0)$ and $g(t) - g(0)$ and add $g(0) + f(0)$ to the obtained result.

2.1.6 Min-Plus Deconvolution

The dual operation of the min-plus convolution is the *min-plus deconvolution* defined as follows for $f, g \in \mathcal{F}$[6].

$$\boxed{(f \oslash g)(t) = \sup_{u \geq 0} \{f(t+u) - g(u)\}} \qquad \text{(Min-Plus Deconvolution)}$$

If one of the function is infinite for some t, the min-plus deconvolution is undefined.

In contrast to the min-plus convolution, the min-plus deconvolution is not closed in $\mathcal{F}$ (the result is not necessarily 0 for $t \leq 0$), not commutative and not associative [BT12, p. 122]. The min-plus deconvolution of $\gamma_{r,b}$ by $\beta_{R,T}$[7] is shown in Figure 2.5.

The min-plus deconvolution enjoys the following properties [BT12, pp. 123, 129].

- **Isotonicity.**
  $\forall f, g, h \in \mathcal{F}, \quad f \leq g \;\Rightarrow\; f \oslash h \leq g \oslash h, \;\; h \oslash f \geq h \oslash g$

[6] The *supremum* $\sup\{\mathcal{S}\}$ of a set $\mathcal{S}$ is defined as the smallest upper bound of $\mathcal{S}$. For example $\sup\{[0,5[\} = 5$. By convention, $\sup\{\varnothing\} = -\infty$.
The *maximum* $\max\{\mathcal{S}\}$ of a set $\mathcal{S}$ is the element of $\mathcal{S}$ which is bigger than all its other elements. It does not always exist. For example, $[0,5[$ has no maximum. However, we have $\sup\{[0,5]\} = \max\{[0,5]\} = 5$.

[7] These functions are defined respectively in Sections **??** and 5.2.

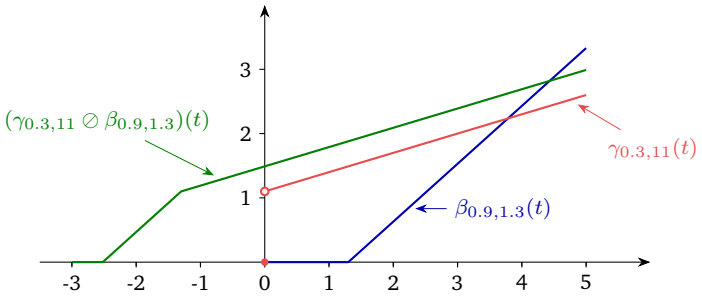


Figure 2.5: Min-plus deconvolution of $\gamma_{r,b}$ by $\beta_{R,T}$.

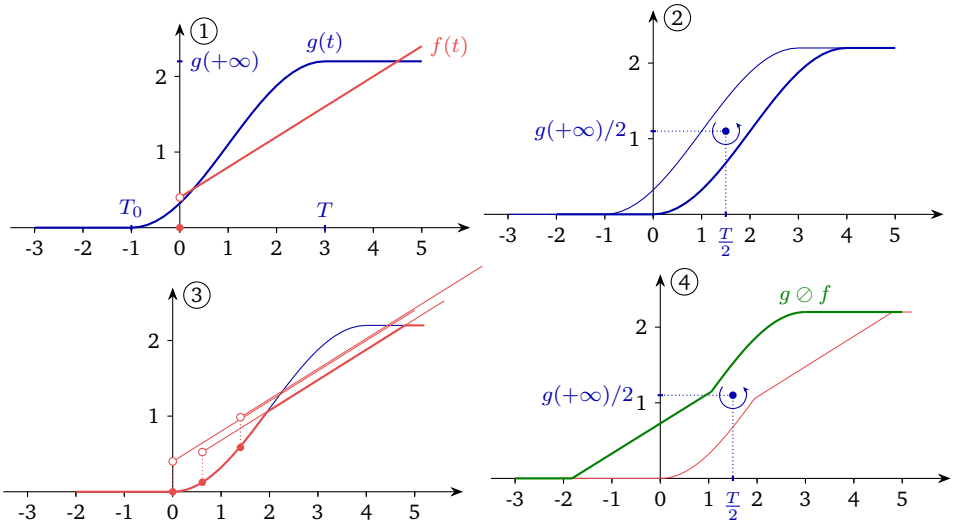


Figure 2.6: Representation of the min-plus deconvolution $g \oslash f$ by time-inversion (adapted from [BT12, p. 127]). g is first rotated around the point $\left(\frac{T}{2}, \frac{g(+\infty)}{2}\right)$ (2). Then, the result is convolved with f (3). Finally, the result of the convolution is rotated back around $\left(\frac{T}{2}, \frac{g(+\infty)}{2}\right)$ to give the final result (4).

- **Composition.**
  $\forall f, g, h \in \mathcal{F}, \quad (f \oslash g) \oslash h = f \oslash (g \otimes h)$
- **Composition with $\otimes$.**
  $\forall f, g \in \mathcal{F}, \quad (f \otimes g) \oslash g \leq f \otimes (g \oslash g)$
- **Addition of a constant.**
  $\forall f, g \in \mathcal{F}, K \in \mathbb{R}^+, \quad (f + K) \oslash g = (f \oslash g) + K$

The min-plus convolution and min-plus deconvolution are said *dual* from each other because they satisfy [BT12, p. 123]

$$f \oslash g \leq h \quad \Leftrightarrow \quad f \leq h \otimes g. \tag{2.6}$$

The min-plus deconvolution of a function $g \in \mathcal{G}$ with finite lifetime by $f \in \mathcal{F}$ can be easily graphically computed. A function g is said to have a finite lifetime if there

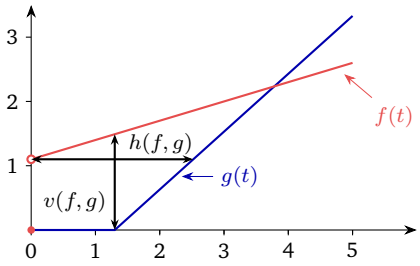


Figure 2.7: Horizontal and vertical deviation between two curves in $\mathcal{F}$.

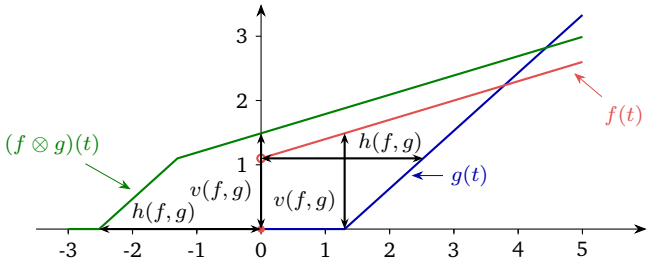


Figure 2.8: Relation between $(f \oslash g)(t)$ and $h(f,g)$ and $v(f,g)$.

exist some finite T_0 and T such that $g(t) = 0$ for $t \leq T_0$ and $g(t) = g(T)$ for $t \geq T$. If we call $\hat{\mathcal{G}}$ the subset of functions of $\mathcal{G}$ with finite lifetime, we can show that, for $f \in \mathcal{F}$ such that $\lim_{t\to+\infty} f(t) = +\infty$ and $g \in \hat{\mathcal{G}}$, we can compute $g \oslash f$ by rotating g by 180° around $\left(\frac{T}{2}, \frac{g(+\infty)}{2}\right)$, computing the min-plus convolution with f and then rotating back again around $\left(\frac{T}{2}, \frac{g(+\infty)}{2}\right)$ [BT12, pp. 125-126]. This is called the representation of the min-plus deconvolution by *time-inversion*. The process is illustrated in Figure 2.6.

2.1.7 Vertical and Horizontal Deviations

The *vertical deviation* and *horizontal deviation* between two curves in $\mathcal{F}$ are two quantities that are very often used in network calculus. They are defined as follows [BT12, p. 128].

$$\boxed{h(f,g) = \sup_{t\geq 0}\{\inf_{d\geq 0}\{d : f(t) \leq g(t+d)\}\}} \qquad \text{(Horizontal Deviation)}$$

$$\boxed{v(f,g) = \sup_{t\geq 0}\{f(t) - g(t)\}} \qquad \text{(Vertical Deviation)}$$

Both quantities are shown in Figure 2.7. It is important to note that these quantities are *not* symmetric with respect to f and g. Graphically, $h(f,g)$ corresponds to the greatest horizontal distance between the graphs of f and g *when* f is left to g and $v(f,g)$ corresponds to the greatest vertical distance between the graphs of f and g *when* f is greater than g.

The min-plus deconvolution allows to easily express these two quantities. Indeed, we can show that [BT12, p. 128]

$$v(f,g) = (f \oslash g)(0), \tag{2.7}$$

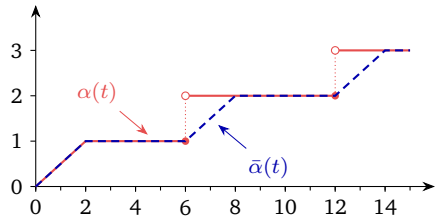


Figure 2.9: Example of an α curve and its sub-additive closure $\bar{\alpha}$ (adapted from [BT12, p. 14]).

$$h(f,g) = \inf_{d \geq 0} \{d : (f \oslash g)(-d) \leq 0\}. \tag{2.8}$$

This is illustrated in Figure 2.8.

The horizontal deviation can also be written as [BT12, p. 128]

$$h(f,g) = \sup_{t \geq 0} \{g^{-1}(f(t)) - t\} = (g^{-1}(f) \oslash t)(0). \tag{2.9}$$

2.1.8 Sub-Additivity

Sub-additive functions are another class of functions that are important in network calculus. A function $f \in \mathcal{F}$ is said sub-additive if and only if [BT12, p. 116]-[Ros50, p. 227]

$$\boxed{\forall s,t \geq 0, \quad f(t+s) \leq f(t) + f(s)}. \tag{Sub-Additivity}$$

The concept is illustrated in Figure 2.9. The α curve is not sub-additive while the $\bar{\alpha}$ curve is.

Letting $t = t' - s$, one can easily see that an equivalent definition is

$$f \leq f \otimes f. \tag{2.10}$$

If $f(0) = 0$, we know that $f \geq f \otimes f$ and it is hence equivalent to imposing that $f = f \otimes f$.

It can be shown that the sum and min-plus convolution of two sub-additive functions are also sub-additive [BT12, pp. 117-118].

A lower-star-shaped function passing through the origin is sub-additive [BT12, p. 117]. Hence, if $f(0) = 0$, we have

$$f \text{ is concave } \Rightarrow f \text{ is lower-star-shaped } \Rightarrow f \text{ is sub-additive}, \tag{2.11}$$

but none of the reverse implications holds [BO62, p. 1207].

If $f \in \mathcal{F}$, $f \oslash f$ is a sub-additive function of $\mathcal{F}$ passing through the origin [BT12, p. 123].

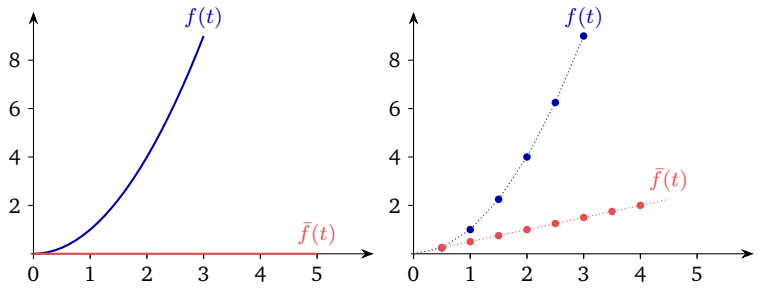


Figure 2.10: The sub-additive closure of $f = t^2$ for $t \geq 0$ is different depending on whether t is continuous (left) or discrete (right) (adapted from [BT12, p. 122]).

Defining the *sub-additive closure* $\bar{f}$ of a function $f \in \mathcal{F}$ as[8] [BT12, p. 118]

$$\boxed{\bar{f} = \delta_0 \wedge f \wedge (f \otimes f) \wedge (f \otimes f \otimes f) \wedge \ldots \triangleq \inf_{n \geq 0} \{f^{(n)}\}}, \qquad \text{(Sub-additive Closure)}$$

it can be shown that $\bar{f}$ is the greatest sub-additive function smaller than f and passing through the origin [BT12, pp. 119-120]. From this, we can easily show that, for $f \in \mathcal{F}$, [BT12, pp. 120, 125]

$$\boxed{f(0) = 0 \text{ and } f \text{ is sub-additive} \quad \Leftrightarrow \quad f \otimes f = f \quad \Leftrightarrow \quad \bar{f} = f \quad \Leftrightarrow \quad f \oslash f = f}. \qquad (2.12)$$

Figure 2.9 shows a curve α and its sub-additive closure $\bar{\alpha}$.

The sub-additive closure also enjoys the following properties [BT12, p. 120].

- **Isotonicity.**
  $\forall f, g \in \mathcal{F}, \quad f \leq g \;\Rightarrow\; \bar{f} \leq \bar{g}$
- $\forall f, g \in \mathcal{F}, \quad \overline{f \wedge g} = \bar{f} \otimes \bar{g}$
- $\forall f, g \in \mathcal{F}, \quad \overline{f \otimes g} \geq \bar{f} \otimes \bar{g}$
- $\forall f, g \in \mathcal{F} : f(0) = g(0) = 0, \quad \overline{f \otimes g} = \bar{f} \otimes \bar{g}$

Figure 2.10 shows that choosing t continuous or discrete may have an impact on the sub-additive closure of a function.

2.2 Max-Plus Algebra

Max-plus algebra consists in replacing, in min-plus algebra, the infimum operator by the supremum operator ($\vee$) and $+\infty$ by $-\infty$ in the set of elements on which operations are performed. This produces the algebraic structure $(\mathbb{R} \cup \{-\infty\}, \vee, +)$. This structure is also a commutative dioid [BT12, p. 129]. As we will see in the following Sections, max-plus algebra derives similar properties as min-plus algebra.

[8] δ_T is defined on page 26.

2.2.1 Max-Plus Convolution

The definition of the *max-plus convolution* is obtained by replacing the infimum operator by a supremum in the definition of the min-plus convolution. For $f, g \in \mathcal{F}$, we have

$$\boxed{(f \overline{\otimes} g)(t) = \sup_{0 \leq s \leq t} \{f(t-s) + g(s)\}}. \qquad \text{(Max-Plus Convolution)}$$

(If $t < 0$, $(f \overline{\otimes} g)(t) = 0$).

2.2.2 Max-Plus Deconvolution

Similarly, the *max-plus deconvolution* is defined, for $f, g \in \mathcal{F}$, as

$$\boxed{(f \overline{\oslash} g)(t) = \inf_{u \geq 0} \{f(t+u) - g(u)\}}. \qquad \text{(Max-Plus Deconvolution)}$$

2.2.3 Linearity of Min-Plus Deconvolution

We can show that the min-plus deconvolution is linear in the max-plus algebra. Indeed, it enjoys the following property [BT12, p. 129]:

$$\forall f, g, h \in \mathcal{F}, \quad (f \vee g) \oslash h = (f \oslash h) \vee (g \oslash h). \tag{2.13}$$

2.2.4 Super-Additivity

Super-additive functions in max-plus algebra are the equivalent of sub-additive functions in min-plus algebra. The definition can be obtained by changing the $\leq$ into a $\geq$ in the definition of sub-additivity. Hence, a function $f \in \mathcal{F}$ is said super-additive if and only if [MOA11, p. 650]-[BO62, p. 1203]

$$\boxed{\forall s, t \geq 0, \quad f(t+s) \geq f(t) + f(s)}. \qquad \text{(Super-additivity)}$$

Letting $t = t' - s$, one can easily see that

$$f \geq f \overline{\otimes} f \tag{2.14}$$

is an equivalent definition. It is also equivalent to imposing that $-f$ is sub-additive [MOA11, p. 650].

It can be shown that if f and g are star-shaped (resp. super-additive), then $af + bg$ ($a \geq 0,\ \ b \geq 0$) is also star-shaped (resp. super-additive) [BO62, p. 1206].

A star-shaped function passing through the origin is super-additive [MOA11, p. 650]. Hence, if $f(0) = 0$, we have

$$f \text{ is convex } \Rightarrow f \text{ is star-shaped } \Rightarrow f \text{ is super-additive}, \tag{2.15}$$

but none of the reverse implications holds [BO62, p. 1207].

CHAPTER 3

DATA MODELING

3.1 Data Flow

A data flow is represented by a *cumulative function* $R(t)$ representing the number of bits seen on the flow during the time interval $[0, t]$. $R(t)$ is wide-sense increasing and we assume that $R(t) = 0 \ \ \forall t \leq 0$. Hence, $R(t) \in \mathcal{F}$.

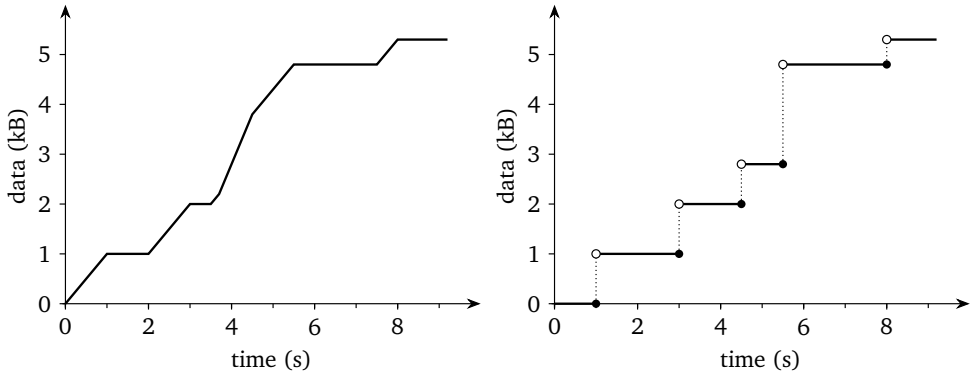
Examples of $R(t)$ curves are shown in Figure 3.1. Different models can be used. Both Figures represent the same flow arriving at a given node. The flow is an aggregation of two flows coming from two different input links. From the first link, packets of length 1 kB, 0.8 kB and 0.5 kB are sent respectively after 0, 3.7 and 7.5 seconds. From the second link, packets of length 1 kB and 2 kB are sent respectively after 2 and 3.5 seconds. Both links have a transmission rate of 1 kB/s. In Figure 3.1a, both data and time are continuous. This is called the *fluid model*. Packets are transmitted bit-by-bit. In Figure 3.1b, time is continuous but data is discrete. Packets are considered seen only when fully received. This is the *general continuous time model*. We have to take the convention that R is either left- or right-continuous[1]. Here, we represented a left-continuous function. We will keep this convention throughout the document.

A data flow is also sometimes represented by its *rate function*, which corresponds to the derivative of the $R(t)$ function in the fluid model [Cru91, pp. 115-116]. On a single link, the rate function can take only two values: zero when the link is idle and the transmission rate of the link when the link is used [Cru91, p. 116]. The rate function can however take mores values (but still from a finite set) if we observe data arriving at an aggregation point. This is the case in Figure 3.1a, in which the derivate of $R(t)$ takes only three different values: 0 kB/s when none of the incoming links is sending, 1 kB/s when only one of the incoming link is sending and 2 kB/s when both incoming links are sending.

3.2 System

A *system* $\mathcal{S}$ (as shown in Figure 3.2) is defined as blackbox taking data $R(t)$ (the *input function*) as input and delivering the data $R^*(t)$ (the *output function*) at its output after a variable delay. A system might represent a single buffer, a complex communication node or even a complete network. The analyzed communication network can then be represented as the interconnection of these systems, also called *network elements* [Cru91, p. 115].

[1] Informally, a function is *left-continuous* (resp. *right-continuous*) if no jump occurs when the limit point is approached from the left (resp. right) [AE10, p. 79].



(a) Fluid model. (b) General continuous time model.

Figure 3.1: Example of cumulative functions $R(t)$ representing data arriving at an aggregation node. From a first link, packets of length 1 kB, 0.8 kB and 0.5 kB are sent respectively after 0, 3.7 and 7.5 seconds. From a second link, packets of length 1 kB and 2 kB are sent respectively after 2 and 3.5 seconds. Both links have a capacity of 1 kB/s. In the fluid model, data is observed bit-by-bit. In the general continuous time model, data is observed packet-by-packet.

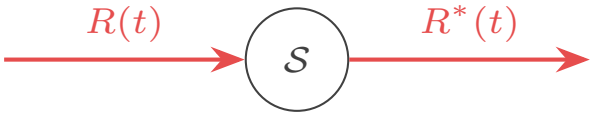
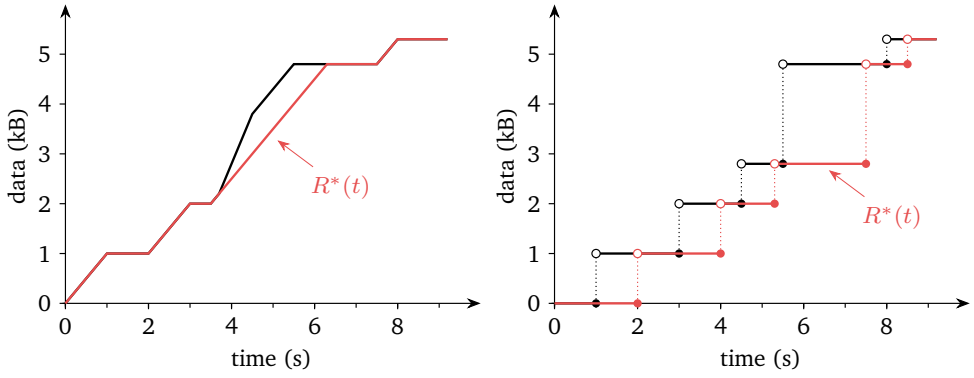
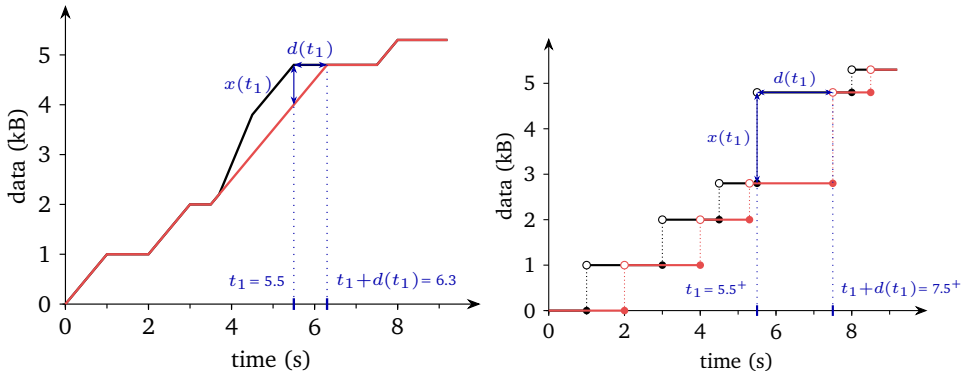


Figure 3.2: A network calculus system takes data $R(t)$ as input and delivers the data $R^*(t)$ at its output.



(a) Fluid model. (b) General continuous time model.

Figure 3.3: Example of input and output functions for a system serving input data at a rate of 1 kB/s.



(a) Fluid model.

(b) General continuous time model.

Figure 3.4: Graphical representation of the backlog and delay experienced by the last bit of the packet sent after 3.5 seconds. In the fluid model, the backlog and delay are smaller because the node does not have to wait for the entire packet to be received to start sending it.

For a system which serves data at a rate of 1 kB/s, Figure 3.3 shows the output functions $R^*(t)$ corresponding to the input functions shown in Figure 3.1. In the fluid model (Figure 3.3a) this results in sending bits as soon as they arrive at a rate of 1 kB/s. In the general continuous time model (Figure 3.3b) this results in serving a packet when it has been fully received and considering it out of the system when it has been fully sent.

3.3 Backlog and Virtual Delay

Two interesting quantities can be derived from the input and output functions [BT12, p. 5]. The *backlog* $x(t)$ corresponds to the amount of data held inside the system at time t.

$$\boxed{x(t) = R(t) - R^*(t)} \qquad \text{(Backlog)}$$

The *virtual delay* $d(t)$ corresponds to the delay that would be experienced by a bit at time t if all bits received before it are served before it[2].

$$\boxed{d(t) = \inf_{\tau \geq 0} \{\tau : R(t) \leq R^*(t+\tau)\}} \qquad \text{(Virtual Delay)}$$

Figure 3.4 shows the graphical interpretation of these two quantities. The virtual delay (resp. the backlog) at time t corresponds to the horizontal deviation (resp. vertical deviation) between the input and output curves starting from the point $(t, R(t))$.

Interestingly, the Figure shows that the backlog and virtual delay can differ from one model to the other. Indeed, the delay experienced by the last bit of the 2 kB packet is different. For the fluid model, in Figure 3.4a, the last bit of the 2 kB packet enters the system at time $t_1 = 5.5$ and leaves it at time $t_1 + d(t_1) = 6.3$. The delay experienced by this bit[3] is hence $d(t_1) = 0.8$. For the general continuous time model, in Figure 3.4b,

[2] That is the reason why it is called *virtual* delay.

[3] The virtual delay corresponds to the delay because bits/packets leave the system in the same order as they entered it.

the last bit of the 2 kB packet enters the system at time[4] $t_1 = 5.5^+$. With similar computations, we obtain a virtual delay (equal to the delay) of $d(t_1) = 2$. This is of course in accordance with the assumptions of each model. In the general continuous time model, the node must wait for the entire packet to be received to start sending it, which induces a delay. This can also be seen for all other packets. For example, the two first packets experience a delay (and generate a backlog) in the general continuous time model but not in the fluid model. This delay corresponds to the time needed to wait for the complete packet to be fully received and fully sent.

Since most of the communication networks are nowadays packet-based, we are only interested in working with the general continuous time model. Indeed, a forwarding device starts forwarding a packet to an output link only when this packet is fully received. However, it is often easier to use the fluid model. This is not a problem. We will see in Chapter 7 how it is possible to switch from the fluid model (bit-by-bit) to the general continuous time model (packet-by-packet).

[4] $t^+ = \inf_{x \in \mathbb{R}} \{x > t\}$.

CHAPTER 4

ARRIVAL CURVES

Obviously, in order to provide performance bounds for a flow, we must have a lower bound of the service the network can offer and an upper bound of the flow characteristics. In other words, we need to know the minimum service a network guarantees to offer and the maximum amount of data a flow will send. These are respectively given by *service curves* (defined in Chapter 5), which model the network and its nodes, and by *arrival curves* (defined in this Chapter), which model flows.

4.1 Definition

A wide-sense increasing function α is an *arrival curve* for a flow R if and only if [BT12, p. 7]-[Cru91, p. 116]

$$\boxed{\forall s \leq t, \;\; R(t) - R(s) \leq \alpha(t-s)}. \qquad \text{(Arrival Curve)}$$

We also say that the flow R is α-*smooth*.

Figure 4.1 illustrates the concept. The arrival curve constraint means that, during any time window of width τ, the amount of additional data sent by the flow is limited by $\alpha(\tau)$. Because it limits the allowed burstiness of traffic, it is also called a *burstiness constraint* [Cru91, p. 114].

Graphically, if we draw instances of the arrival curve starting at any point of the $R(t)$ curve, $R(t)$ must always remain smaller than all these instances.

From the definition of the min-plus convolution, we can easily show that the definition of an arrival curve is equivalent to [BT12, p. 15]

$$R \leq R \otimes \alpha. \tag{4.1}$$

From this other definition and from the isotonicity and associativity of $\otimes$, we can also easily show that if α_1 and α_2 are arrival curves for a flow, then so is $\alpha_1 \otimes \alpha_2$ [BT12, p. 15].

It can be shown that we can always reduce an arrival curve to be left-continuous. More precisely, an arrival curve $\alpha(t)$ can always be reduced to $\alpha_l(t) = \sup_{s<t} \alpha(s)$ without changing the set of flows for which it is an arrival curve [BT12, p. 9]. $\alpha_l(t)$ corresponds to the limit to the left of $\alpha(t)$ and is left-continuous. Since we have $\alpha_l(t) \leq \alpha(t)$, this is always a better bound for the flow.

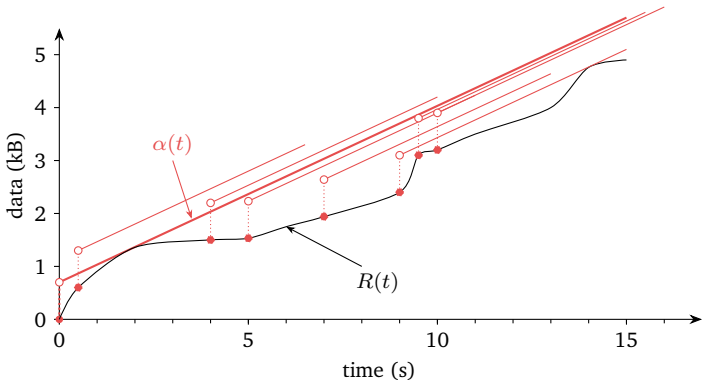


Figure 4.1: Graphical illustration of the arrival curve concept (adapted from [BT12, p. 7]). If we draw instances of the arrival curve starting at any point of the $R(t)$ curve, $R(t)$ must always remain smaller than all these instances.

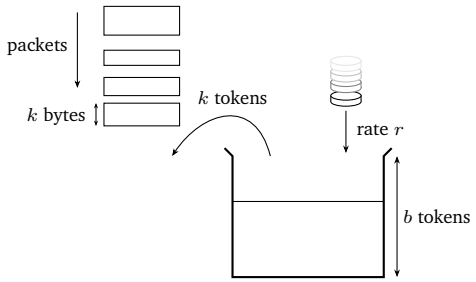


Figure 4.2: Illustration of the token bucket algorithm. When a packet of size k has to be transmitted, it removes k tokens from the token bucket. The packet is declared non-conformant if there are not enough tokens in the bucket.

If two flows R_1 and R_2 have respectively α_1 and α_2 as arrival curves, then their aggregate $R_1 + R_2$ has $(\alpha_1 + \alpha_2)$ as arrival curve [Cru91, p. 116].

Affine arrival curves are the most commonly used arrival curves.

$$\gamma_{r,b}(t) = \begin{cases} rt + b & \text{if } t > 0 \\ 0 & \text{otherwise} \end{cases} \qquad \text{(Affine Arrival Curve)}$$

b is called the *burst* and r the *rate*. This kind of arrival curve allows a source to send b bits at once, but not more than r b/s over the long run. The arrival curve shown in Figure 4.1 belongs to this family.

4.2 Token Bucket and Leaky Bucket Algorithms

An affine arrival curve is closely related to the concepts of *token bucket* and *leaky bucket*. For this reason, arrival curves belonging to this family are also sometimes called token bucket arrival curves or leaky bucket arrival curves. Both concepts define an algorithm to determine if some data is conformant or non-conformant to some traffic policy.

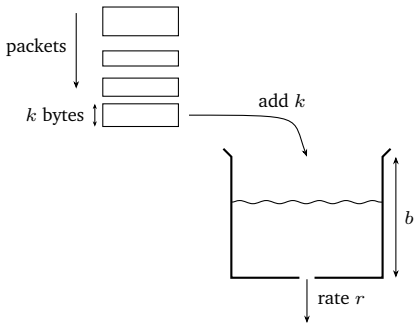


Figure 4.3: Illustration of the leaky bucket algorithm. When a packet of size k has to be transmitted, it adds k units of fluid in the leaky bucket. The packet is declared non-conformant if this results in the bucket to overflow.

The *token bucket* algorithm [TW11, pp. 407-411] is illustrated in Figure 4.2. A bucket of size b tokens, initially full, is filled at a rate of r tokens per second. If the bucket is full, no tokens are added anymore. When a packet of size k has to be transmitted into the network, it must remove k tokens from the bucket. If there are not enough tokens, the packet is declared non-conformant. Non-conformant data can be either queued or dropped. If there are enough tokens, k tokens are removed and the packet can be transmitted.

The *leaky bucket* algorithm [BT12, p. 10]-[TW11, pp. 407-411]-[ELL90, p. 203]-[Cru91, p. 115] is illustrated in Figure 4.3. A bucket of size b, initially empty, leaks fluid at a rate of r units of fluid per second when it is not empty. When a packet of size k has to be transmitted into the network, it must add k units of fluid into the bucket. A packet that would cause the bucket to overflow is declared non-conformant (and is then queued or dropped). Otherwise, the fluid is added and the packet is transmitted.

Note that the tokens and the fluid do not represent bits. They, in fact, play a role similar to a police officer at an intersection with a stopwatch.

Whereas these two formulations are different, it can be easily seen that they are equivalent. They both limit the long-term rate of a flow to r b/s but allow bursts of up to b bytes. The analogy with affine arrival curves is then obvious. A leaky bucket with leak rate r and bucket size b, or a token bucket with filling rate r and bucket size b, forces a flow to be constrained by the arrival curve $\gamma_{r,b}$ [BT12, p. 11].

Attention must be paid that another leaky bucket algorithm, which is *not* equivalent to the one here above, is described in the literature. We will refer to this algorithm as the *leaky bucket as a queue algorithm*. In this algorithm [Tan03, pp. 303-305], when a packet has to be transmitted, it is added in the bucket. If there is not enough place for it, it has to be discarded. Then, the bucket transmits the packets that it contains at a rate of r b/s. The main difference with the leaky bucket algorithm described above is that the leaky bucket as a queue algorithm enforces a rigid output pattern at the maximum rate r whereas the leaky bucket algorithm allows bursts to be transmitted.

Because of the leaky bucket as a queue algorithm, leaky buckets and token buckets are sometimes considered to be different, like in [Tan03, p. 306]. However, most of the

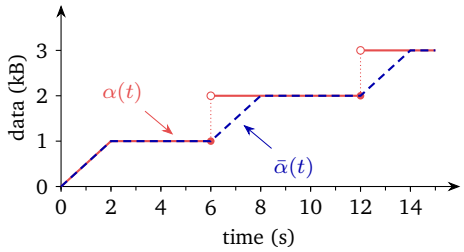


Figure 4.4: Illustration of the *good* arrival curve concept (adapted from [BT12, p. 14]). α can always be replaced by its sub-additive closure $\bar{\alpha}$.

modern literature considers the leaky bucket and token bucket algorithms as we have described them here, therefore considering that leaky buckets and token buckets are equivalent and even using their names interchangeably [KR13, pp. 645-647]-[TW11, pp. 407-411]-[BT12, p. 10].

4.3 Good Arrival Curves

Sometimes, the sheer knowledge that a flow R is constrained by an arrival curve α allows to obtain a better bound than α itself. This is illustrated in Figure 4.4. $\alpha(t)$ for $t \leq 2$ tells us that, during a 2 seconds time window, data cannot arrive faster than 0.5 kB/s. We have $\alpha(6) = 1$, which allows R to send maximum 1 kB of data in a 6 seconds time frame. We also have $\alpha(7) = 2$, which allows R to send maximum 2 kB of data in a 7 seconds time frame. However, since data sent during a 1 second time frame is limited to $\alpha(1) = 0.5$ bytes and since data sent during a 6 seconds time frame is limited to 1 kB, the data sent during a 7 seconds time frame is implicitly limited to 1.5 kB, which is lower than the bound announced by $\alpha(t)$ itself. The reasoning we had is that a good arrival curve should be such that $\alpha(t+s) \leq \alpha(t) + \alpha(s) \ \ \forall s, t \geq 0$. Indeed, if it is not the case, $\alpha(s) + \alpha(t)$ is a better bound than $\alpha(s+t)$. This inequation corresponds to the definition of sub-additivity.

Therefore, we say that an arrival curve $\alpha \in \mathcal{F}$ is a *good* arrival curve if it is sub-additive and if $\alpha(0) = 0$. We add the *free* condition[1] $\alpha(0) = 0$ to be able to use the equivalences in Equation 2.12 to characterize good arrival curves. Hence, $\alpha \in \mathcal{F}$ is a *good* arrival curve if [BT12, p. 14]

$$\boxed{\alpha(0) = 0 \text{ and } \alpha \text{ is sub-additive } \Leftrightarrow \alpha \otimes \alpha = \alpha \Leftrightarrow \bar{\alpha} = \alpha \Leftrightarrow \alpha \oslash \alpha = \alpha}.$$

(Good Arrival Curve)

From Equation 2.11, we see that any concave or lower-star-shaped function $f(t)$ such that $f(0) = 0$ is a good arrival curve (but not all good arrival curves are concave or lower-star-shaped, e.g. $\bar{\alpha}$ in Figure 4.4).

We can then prove that we can always replace a wide-sense increasing arrival curve α by its sub-additive closure $\bar{\alpha}$ without changing the set of flows for which it is an arrival curve [BT12, p. 15]. For the example described here above, Figure 4.4 shows the sub-additive closure $\bar{\alpha}$ of α.

[1] We say that this is a free condition because, from the definition of an arrival curve, we can always set $\alpha(0) = 0$ for any arrival curve $\alpha \in \mathcal{F}$ without changing the set of flows for which α is an arrival curve. Therefore this does not effectively restrict the set of good functions.

As we can expect, token bucket arrival curves are good arrival curves.

We also said that an arrival curve can be replaced by its limit to the left. How is this related to the sub-additive closure? Actually, the sub-additive closure and limit to the left operations do not commute [BT12, p. 16]. However, we can show that $(\bar{\alpha})_l$ is always a good arrival curve. Therefore, taking the sub-additive closure of α and then the limit to the left will always lead to a good function that is left-continuous, i.e. a *very good* function [BT12, p. 16].

4.4 Minimum Arrival Curve

It can be shown that, for a given flow $R(t)$ with $R(0) = 0$, the source has R as an arrival curve if and only if R is a good function [BT12, p. 16]. The source of the flow is then called a *greedy source*.

We might then wonder if it is possible to get the minimum arrival curve corresponding to a flow R if R is not a good function. Actually, it can be easily shown, from the definition of the min-plus deconvolution, that

$$\boxed{\alpha = R \oslash R} \qquad \text{(Minimum Arrival Curve)}$$

is the *minimum arrival curve* for the flow R and that it is a good arrival curve [BT12, p. 16]. This is also sometimes called the *canonical arrival curve* for R [BJT09, p. 5].

CHAPTER 5

SERVICE CURVES

After having formalized an upper bound of the flow characteristics in the previous Chapter, we will now define a lower bound of the service the network can offer. The details of how packets are handled by a node or network are abstracted using the concept of *service curve*, which we will define in the current Chapter. The combination of arrival and service curves will then be used in the next Chapters to compute performance bounds.

5.1 Definition

A system $\mathcal{S}$, with input and output functions R and R^*, offers a service curve β to R if and only if β is wide-sense increasing, $\beta(0) = 0$ and [BT12, p. 19]

$$\boxed{R^* \geq R \otimes \beta}, \qquad \text{(Service Curve)}$$

or, alternatively,

$$R^*(t) \geq \inf_{s \leq t} \{R(s) + \beta(t-s)\}. \qquad (5.1)$$

Figure 5.1 illustrates this definition. Given R and the service curve β offered by a system $\mathcal{S}$, we can compute $R \otimes \beta$. The output R^* of the system $\mathcal{S}$ will lie in the area (shaded in the Figure) between[1] R and $R \otimes \beta$. An example of possible output is also shown in the Figure.

[1] For causality reasons, we must also have $R^* \leq R$.

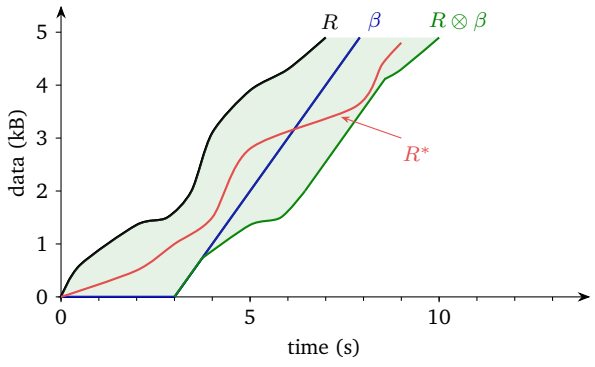


Figure 5.1: Graphical illustration of the service curve concept (adapted from [BT12, p. 19]). The output of the system must always be greater than the convolution of the input with the service curve of the system.

We can see that this service curve concept indeed provides a lower bound on the service a system $\mathcal{S}$ can offer. For example, if a system $\mathcal{S}$ pretends to offer a service curve β, we can be sure that, if we provide R at the input of $\mathcal{S}$, its output will be at least $R \otimes \beta$.

5.2 Common Service Curves

Burst-delay and *rate-latency* functions are the functions most commonly used as service curves. They are defined as follows.

$$\boxed{\delta_T(t) = \begin{cases} +\infty & \text{if } t > T \\ 0 & \text{otherwise} \end{cases}} \qquad \text{(Burst-Delay Function)}$$

$T > 0$ is called the *delay*. As convolving a function with δ_T shifts it by T to the right, this kind of service curve is used to model a first-in-first-out (FIFO) node imposing a delay $d \leq T$ to bits passing through it.

$$\boxed{\beta_{R,T}(t) = R[t-T]^+ = \begin{cases} R(t-T) & \text{if } t > T \\ 0 & \text{otherwise} \end{cases}} \qquad \text{(Rate-Latency Function)}$$

$T \geq 0$ is called the *delay* and $R \geq 0$ the *rate*. This type of service curve is usually used to model an approximation of a generalized processor sharing (GPS) node (see Section 8.2.3). Bits might have to wait up to T before being served with a rate greater or equal to R. If $T = 0$, this models a perfect GPS node allocating a rate of at least R to a flow. The service curve shown in Figure 5.1 is a rate-latency service curve.

5.3 Other Classes of Service Curves

The service curve concept defined in Section 5.1 is the classical service curve concept used in network calculus. Nevertheless, other types of service curves can be defined. We introduce the two most common in the following Sections. [BJT09] provides a more detailed treatment of service curves in network calculus.

In the following, the naming *service curve* corresponds to the concept defined in Section 5.1. In case of ambiguity with the new concepts defined in the two next Sections, the concept defined in Section 5.1 is alternatively referred to as *classical service curve*.

5.3.1 Strict Service Curve

A system $\mathcal{S}$ offers a *strict service curve* β to a flow if, during any backlogged period[2] of duration τ, the output of the system is at least equal to $\beta(\tau)$ [BT12, p. 21]. Mathematically[3] [BJT09, p. 6]

$$\boxed{\forall \text{ backlogged period }]s,t], \quad R^*(t) - R^*(s) \geq \beta(t-s)}. \qquad \text{(Strict Service Curve)}$$

[2] A backlogged period is an interval of time I (can be closed, semi-closed or open) during which the backlog is non-null, i.e. $\forall u \in I : \ R(u) - R^*(u) > 0$ [BJT09, p. 5].

[3] If β is assumed left-continuous, changing $]s,t]$ to $]s,t[$ in the definition does not change the meaning of it [BJT09, pp. 6-7]. We will keep this assumption in this document.

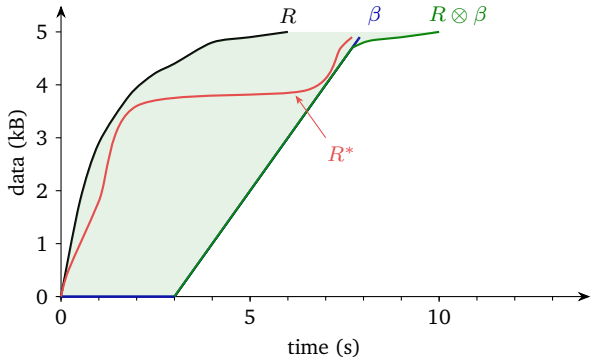


Figure 5.2: The service curve property does not provide guarantees over any interval of time (adapted from [BT12, p. 196]).

The strict service curve property allows to guarantee service during any backlogged interval, which is not possible with the classical service curve property [BT12, p. 196]. Indeed, if for some interval of time, a server gave a service higher than announced by its service curve, the classical service curve property allows the server to be lazy afterwards. This is illustrated in Figure 5.2. The server gave a high service between times 0 and 2 and can hence be lazy (nearly delivering no service) between times 2 and 6.5 though there is some backlogged data in the system (since $R > R^*$). On the other hand, a node offering a strict service curve β guarantees that, for any backlogged period of length τ, it will output at least $\beta(\tau)$ amount of data.

A GPS node is an example of node that offers a strict service curve of the form $\beta = rt$ [BT12, pp. 22, 196] because it will always send the backlogged data at least at the promised rate.

Since both types of service curves represent lower bounds of the service a node offers, $\forall \beta, \beta' \in \mathcal{F}$ if $\beta \leq \beta'$ and if β' is any type of service curve for a system, then β is also a service curve of this type for the system [BJT09, p. 8].

As its name suggests, the strict service curve property is more strict than the service curve property. This means that if a node offers β as a strict service curve to a flow, then it also offers the same curve β as a *classical* service curve to the flow [BT12, p. 22].

5.3.2 Maximum Service Curve

We say that a system $\mathcal{S}$, with input and output functions R and R^*, offers a *maximum service curve* γ to a flow R if and only if $\gamma \in \mathcal{F}$ and [BT12, p. 35]

$$\boxed{R^* \leq R \otimes \gamma}, \qquad \text{(Maximum Service Curve)}$$

or, alternatively,

$$\forall t, s \leq t, \quad R^*(t) \leq R(s) + \gamma(t-s), \tag{5.2}$$

or, also,

$$\forall t, s \leq t, \quad R^*(t) - R^*(s) \leq x(s) + \gamma(t-s), \tag{5.3}$$

where $x(s)$ is the backlog at time s.

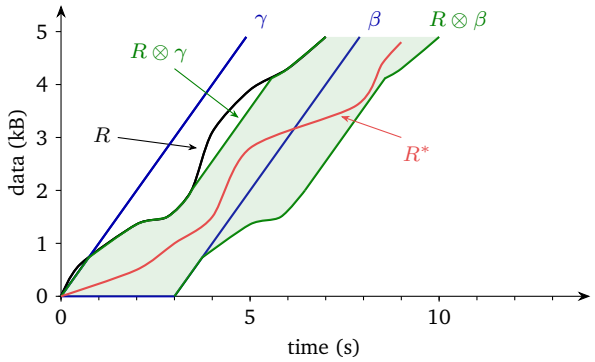


Figure 5.3: Graphical illustration of a maximum service curve coupled with a service curve. The output of the system must always be greater than the convolution of the input by the service curve of the system and lower than the convolution of the input by the maximum service curve of the system.

A given system $\mathcal{S}$ can offer different curves as classical, strict and maximum service curves. For example, Figure 5.3 considers a system $\mathcal{S}$ offering a classical service curve β and a maximum service curve γ. From this, if we provide R at the input of $\mathcal{S}$, we are sure that the output R^* of the system will lie in the area (shaded in the Figure) between $R \otimes \beta$ and $R \otimes \gamma$. An example of possible output is shown in the Figure. The maximum service curve concept, as its name suggests, actually allows a system to provide an *upper* bound on the service it can provide.

It is easy to show that if the output of a lossless node is constrained by σ, then the node offers σ as maximum service curve [BT12, p. 35]. Note that if a node offers σ as maximum service curve, it does not mean that its output has σ as arrival curve (this can be shown with the output bound formula given in Section 6.1.3). This is the property of a shaper (see Section 5.5).

A node offers a maximum service curve of δ_T if and only if it imposes a minimum virtual delay equal to T [BT12, p. 35]. Similarly, if a flow traverses a node with maximum service service γ such that $\gamma(D) = 0$, then the virtual delay $d(t)$ satisfies $d(t) \geq D \ \forall t$ [BT12, p. 36].

5.4 Concatenation

Assuming that a flow traverses systems $\mathcal{S}_1$ and $\mathcal{S}_2$ in sequence, if the systems offer respectively the service curves β_1 and β_2 to the flow, then the concatenation of the two systems offers a service curve $\beta_1 \otimes \beta_2$ to the flow [BT12, p. 28]. Hence, if we note $\beta(\mathcal{S})$ the service curve offered by a system $\mathcal{S}$ and use $\circ$ to denote the concatenation of two systems, we have

$$\boxed{\beta(\mathcal{S}_1 \circ \mathcal{S}_2) = \beta(\mathcal{S}_1) \otimes \beta(\mathcal{S}_2)}. \qquad \text{(Concatenation)}$$

Note that if β_1 and β_2 are strict service curves, $\beta_1 \otimes \beta_2$ is not necessarily strict [BJT09, pp. 13-14]. However, since strict service curves are classical service curves, $\beta_1 \otimes \beta_2$ is a classical service curve for the system. It is actually not possible to define a new

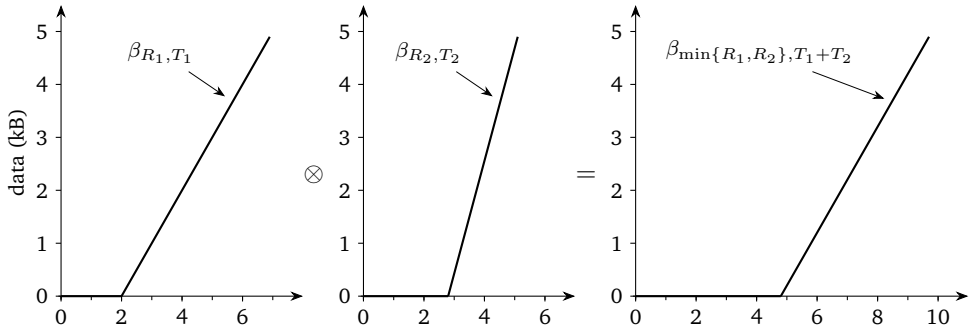


Figure 5.4: The concatenation of two rate-latency service curves results in a rate-latency server whose rate is the minimum of the rates of the two initial servers and whose delay is the sum of the delays of the two initial servers. This can be generalized to any number of rate-latency servers.

notion of service curve which would be preserved through concatenation and which would be stricter than the classical service curve property [BJT09, pp. 14-17].

Figure 5.4 shows that the concatenation of two rate-latency servers β_{R_1,T_1} and β_{R_2,T_2} results in a rate-latency server $\beta_{\min\{R_1,R_2\},T_1+T_2}$ [BT12, p. 28]. This property can easily be generalized to any number n of rate-latency servers β_{R_i,T_i}: their concatenation is a rate-latency server $\beta_{\min\{R_1,\ldots,R_n\},T_1+\ldots+T_n}$.

The concatenation principle is also true for maximum service curves [BT12, p. 35].

5.5 Greedy Shapers

In this Section, we introduce a particular type of network calculus system: *greedy shapers*. As any system, they can be modeled using service curves. However, they enjoy a stronger input-output relationship than service curves which will lead to interesting properties in the next Chapters.

Let us use the same step-by-step definition as [BT12] to define what a greedy shaper is.

Policer A *policer* with curve σ is a device deciding which bits of a flow conform to an arrival curve σ [BT12, p. 30]. Policing can be performed at the boundary of a network to check that users do not send more than specified by some agreement. Non-conformant traffic can then be discarded or marked as low priority or best effort.

Shaper A *shaper* with shaping curve σ is a bit processing device that forces its output to have σ as an arrival curve [BT12, p. 30].

Greedy Shaper A *greedy shaper* with shaping curve σ is a shaper that delays the input bits in a buffer whenever sending them would violate the constraint σ, but outputs them as soon as possible [BT12, p. 30]. This means at the earliest time t such that [Geo+96, p. 482]

$$R(t) - R(t-\tau) \leq \sigma(\tau) \quad \forall \ 0 \leq \tau \leq t.$$

The *greedy* adjective comes from the fact that the device outputs data *as soon as possible*. In contrast, we could have a *lazy shaper* never sending the data.

A lossless greedy shaper with shaping curve σ (and empty at time 0) has an input-output relationship of the form [BT12, pp. 31-32]

$$\boxed{R^* = R \otimes \bar{\sigma}}. \qquad \text{(Greedy Shaper)}$$

Hence, a greedy shaper with a sub-additive shaping curve σ offers σ both as a service curve and a maximum service curve [BT12, pp. 32, 35]. However, it does not satisfy the strict service curve property [BT12, p. 22].

It can be easily shown that if an α-smooth (α being a good function) flow is input to a greedy shaper with shaping curve σ, then the output flow is still α-smooth [BT12, p. 34], i.e. *greedy shapers keep arrival constraints*. Hence, the output of the greedy shaper has $\min\{\alpha, \sigma\}$ as an arrival curve. If σ is also a good function, then the sub-additive closure of $\min\{\alpha, \sigma\}$ is given by $\alpha \otimes \sigma$ [BT12, p. 34] and the output of the shaper is therefore $(\alpha \otimes \sigma)$-smooth.

In min-plus system theory, a system is linear and time invariant (LTI) if its input-output relationship has the form $Y = X \otimes \sigma$. Greedy shapers are hence min-plus LTI systems.

[BT12, pp. 36-38] gives an example showing that the modeling of a network element as a shaper, when possible, provides stronger bounds than the maximum service curve (see Section 6.1 for bounds results).

CHAPTER 6

NETWORK CALCULUS BASICS

Armed with the concepts of arrival and service curves, we are now able to develop the main results of network calculus. After having introduced the mechanisms to compute the worst-case bounds in Section 6.1, we present in Section 6.2 the particular values in commonly encountered cases. We then introduce two important network calculus principles in Sections 6.3 and 6.4.

6.1 Bounds

From the arrival curve α of a flow and the service curve β of a node, network calculus theory allows to compute an upper bound of

- the backlog generated by the flow at this node,
- the virtual delay the flow will experience at the node,
- the new arrival curve α^* of the flow at the output of the node.

This is illustrated in Figure 6.1.

6.1.1 Backlog Bound

From the definitions of service and arrival curves, it can be shown that the backlog $x(t)$ at a node offering a service curve β to a flow with arrival curve α is such that [BT12, pp. 22-23]

$$\boxed{x(t) \leq v(\alpha, \beta)}, \qquad \text{(Backlog Bound)}$$

i.e., is bounded by the vertical deviation between the arrival and service curves.

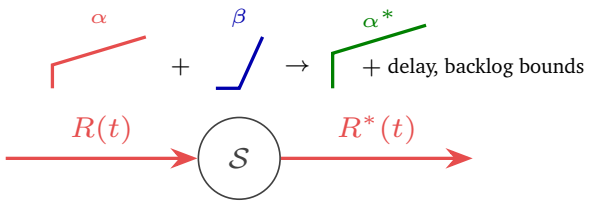


Figure 6.1: From arrival and service curves, network calculus allows to compute bounds on the delay a flow will experience, on the backlog a flow will cause and on the new arrival curve of the flow at the output of the system.

It can be shown that, if α is a good function, β is wide-sense increasing and $\beta(0) = 0$, then the bound is *tight*, i.e. there is a system offering a service curve β to a flow constrained by α and which is achieving the bound. Of course, the bound is as tight as the arrival and service curves are close to the physical flow and system they model. Indeed, it can easily be seen that if we overestimate the arrival curve of the flow or underestimate the service curve of the node, the obtained bounds will be overestimated. Hence, the bound is called a *worst-case* bound and it can be achieved by a greedy source [BT12, p. 27].

6.1.2 Delay Bound

Similarly, it can be shown that the virtual delay $d(t)$ experienced by a flow with arrival curve α at a node offering a service curve β is such that [BT12, p. 23]

$$\boxed{d(t) \leq h(\alpha, \beta)}, \qquad \text{(Delay Bound)}$$

i.e., is bounded by the horizontal deviation between the arrival and service curves.

Similarly to the backlog bound, if α is a good function, β is wide-sense increasing and $\beta(0) = 0$, then the bound is tight and can be achieved by a greedy source [BT12, p. 27].

6.1.3 Output Flow Bound

After having traversed a node, a flow R is transformed into R^*. One might wonder then what could be an arrival curve for R^*. As data of the flow can be buffered at the node before being served, we expect the burst of the flow to increase when traversing a node. We can show that an α-smooth flow traversing a node with service curve β and maximum service curve γ gets out of the node with an arrival curve α^* given by [BT12, pp. 23, 35-36]

$$\boxed{\alpha^* = (\alpha \otimes \gamma) \oslash \beta}. \qquad \text{(Output Flow)}$$

If the node has no maximum service curve, i.e. if $\gamma = \delta_0$, this reduces to

$$\alpha^* = \alpha \oslash \beta. \tag{6.1}$$

Since we expect $\alpha \otimes \gamma$ to be smaller than α, we see that the knowledge of a maximum service curve can lower, i.e. improve, the output bound.

Since traffic exiting a system might be input traffic to a second system, the knowledge of this output bound allows to analyze this second system and hence to perform a network-wide worst-case analysis.

$\alpha^*(0^+)$ corresponds to the maximum amount of data of the flow the node will output in an infinitely small amount of time, i.e. to the new maximum burst of the flow. Since $(\alpha \oslash \beta)(0) = v(\alpha, \beta)$ (see Section 2.1.7), we see that the new maximum burst of the flow corresponds to the maximum backlog that can be observed at the node. This is intuitive. Indeed, all data that accumulates in the node can be released instantly as a burst if the node eventually receives an infinite service, which the service curve concept allows.

Recall that the min-plus deconvolution is not closed in $\mathcal{F}$. Therefore $\alpha^*(t)$ might not be zero for $t \leq 0$. α^* is hence not a good function. However, it is sub-additive. Thus, the modified function $\delta_0 \wedge \alpha^*$ is a good function (even a very good one if α is left-continuous) [BT12, p. 28]. Using the formulas above, the good arrival curve in $\mathcal{F}$ for the output flow can hence be obtained by setting $\alpha^*(t) = 0 \ \forall t \leq 0$.

The bound is tight (i.e. α^* is the *minimum* arrival curve for R^*) if α is a good function and left-continuous, β is wide-sense increasing, $\beta(0) = 0$ and $\alpha \overline{\oslash} \alpha$ (see Section 2.2.2) is not bounded from above [BT12, pp. 27-28, 56-58].

6.1.4 Delay from Backlog

In this Section, we present cases when it is possible to compute the maximum virtual delay at time t based on the backlog that we can measure at this time. This is useful when the arrival curve of a flow is not known but the backlog generated by the flow at the node can be measured, which is usually the case (buffer usage monitoring).

As we already know, if we furthermore assume the node to be FIFO, the bound on the virtual delay corresponds to the maximum delay a bit of the flow will experience at this time.

6.1.4.1 Classical Service Curve Case

It is not possible, in general, to bound virtual delay from backlog using the framework of classical service curves [BT12, p. 38]. However, in the particular case of a node offering a service curve β and a maximum service curve γ satisfying

$$\beta(t) = \gamma(t - v), \tag{6.2}$$

then we have [BT12, p. 38]

$$(\gamma \overline{\oslash} \gamma)(d(t) - v) \leq x(t). \tag{6.3}$$

If, in addition, γ is super-additive, then

$$\beta(d(t)) \leq x(t). \tag{6.4}$$

For example, for a node with service curve $\beta = \beta_{R,T}$ and maximum service curve $\gamma = \beta_{R,T'}$ (with $T' \leq T$), we have [BT12, p. 38]

$$d(t) \leq \frac{x(t)}{R} + T.$$

6.1.4.2 Strict Service Curve Case

The strict service curve guarantee allows to compute the maximum virtual delay at time t based on the backlog $x(t)$ at this time. Indeed, we can show that [BT12, p. 196]

$$d(t) \leq \beta^{-1}(x(t)). \tag{6.5}$$

The difference with the classical service curve case is that we do not require γ to be super-additive and to satisfy Equation 6.2.

6.1.5 Output from Delay

A major drawback of the output bound is that the deconvolution operation is not easy to perform in the general case. [Cru91, p. 117] shows that an output bound α^* can be obtained from the delay bound $\bar{D}$ at a node and is given by

$$\alpha^*(t) = \alpha(t + \bar{D}), \tag{6.6}$$

i.e. that the arrival curve of a flow after a node can be obtained by shifting to the left its initial arrival curve by the maximum delay the flow could experience at this node.

Note that the bound obtained by this formula is not tight. Indeed, we will see in the next Section that using the formula involving the deconvolution operator allows to obtain a better, i.e. a lower, output bound. The latter provides a better bound because it takes advantage of the exact knowledge of the network element structure (its service curve) while the formula proposed here only uses the delay it induces. Nevertheless, Equation 6.6 is useful when either

- the arrival and service curves combination leads to intractable or complicated computation of the tight bound,
- or the maximum delay is known but not the service curve.

6.2 Common Bounds Results

In the two following Sections, we present the bounds values for the two most commonly encountered arrival-service curves combinations.

6.2.1 Leaky Bucket Flow through Rate Latency Server

Considering a flow constrained by a leaky bucket arrival curve $\gamma_{r,b}$ going through a node offering a service curve $\beta_{R,T}$, we obtain the following bounds (for $r \leq R$), shown in Figure 6.2 [BT12, p. 24].

$$x(t) \leq b + rT \tag{6.7}$$

$$d(t) \leq T + \frac{b}{R} \tag{6.8}$$

$$\alpha^*(t) = \gamma_{r,b+rT} \tag{6.9}$$

6.2.2 VBR Flow through Rate Latency Server

We call a *constant bit rate* (CBR) connection a flow constrained by a leaky bucket arrival curve. We then call a *variable bit rate* (VBR) connection a flow constrained by two leaky buckets in series [BT12, p. 13], i.e. by an arrival curve of the type

$$\alpha(t) = \alpha_1(t) \otimes \alpha_2(t) = (M + pt) \otimes (b + rt) = \min\{M + pt, b + rt\}. \tag{6.10}$$

Indeed, both leaky buckets curves are lower-star-shaped and their convolution is therefore their minimum. We assume that $r \leq R$, $M \leq b$ and $p \geq r$. Such an arrival curve is shown in Figure 6.3. The Internet *Integrated Services* (IntServ) framework [RFC1633] uses this family of arrival curve. The 4-tuple (p, M, r, b) is hence often called, from IntServ jargon, a *T-SPEC* (traffic specification) [BT12, p. 13]. Considering

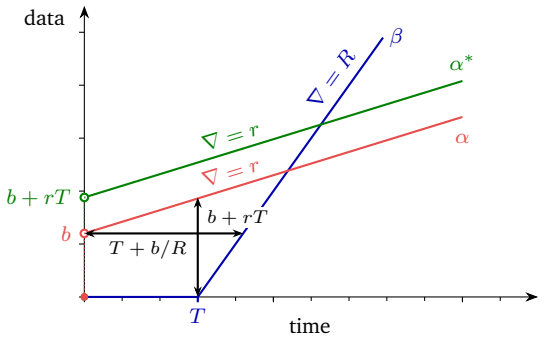


Figure 6.2: Graphical illustration of the computation of the delay, backlog and output bounds for a leaky bucket flow traversing a rate latency server (adapted from [BT12, p. 24]).

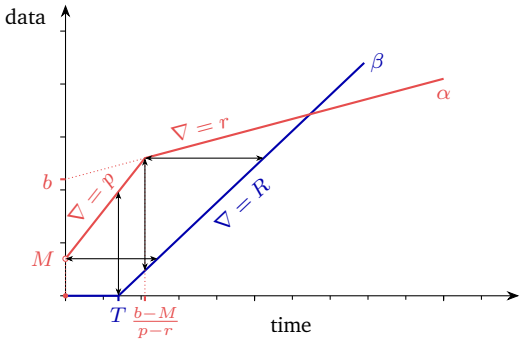


Figure 6.3: Graphical illustration of the computation of the delay and backlog bounds for a VBR flow traversing a rate latency server (adapted from [BT12, p. 25]).

such a flow going through a node offering a service curve $\beta_{R,T}$, we obtain the following bounds (for $r \leq R$), shown in Figures 6.3 and 6.4 [BT12, pp. 24-25].

$$x(t) \leq b + rT + \left(\frac{b-M}{p-r} - T\right)^+ ((p-R)^+ - p + r) \tag{6.11}$$

$$d(t) \leq T + \frac{M + \frac{b-M}{p-r}(p-R)^+}{R} \tag{6.12}$$

$$\alpha^*(t) = \begin{cases} \text{if } \frac{b-M}{p-r} \leq T & b + r(T+t) \\ \text{otherwise} & \min\{b + r(T+t),\ \ (t+T)(p \wedge R) + M + \frac{b-M}{p-r}(p-R)^+\} \end{cases} \tag{6.13}$$

Delay and Backlog Bounds. From the convexity and linearity of the region between α and β, we know that the maximum horizontal and vertical deviations can only be reached at angular points of either α or β. From this, only two values are possible for both the delay and backlog bounds [BT12, p. 24]. These are shown in Figure 6.3. Some max-plus algebra then leads to the formulas given above.

Output Bound. [BT12, p. 25] shows how it is possible to compute the min-plus deconvolution of a concave function α by the rate latency function $\beta_{R,T}$. First, we

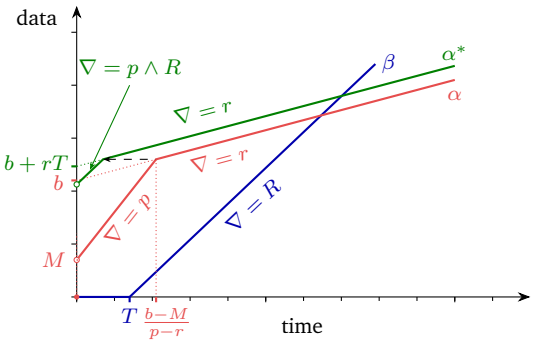


Figure 6.4: Graphical illustration of the computation of the output bound for a VBR flow traversing a rate latency server.

define

$$t_0 = \inf_{t \geq 0} \{\alpha'(t) \leq R\}.$$

In our case, we have

$$t_0 = \begin{cases} \frac{b-M}{p-r} & \text{if } p > R \\ 0 & \text{otherwise} \end{cases}.$$

Then, we can obtain α^* from α by

- replacing α on $[0, t_0]$ by the linear function with slope R that has the same value as α for $t = t_0$,
- shifting by T to the left.

The formula given above for α^* results from this applied to a VBR flow [BT12, p. 25]. In simple words, this formula means that

$$\alpha^*(t) = \begin{cases} \text{if } \frac{b-M}{p-r} \leq T & b + r(T+t) \\ \text{otherwise if } p > R & \alpha \text{ shifted by } T \text{ to the left and replacing first slope } p \text{ by } R \\ \text{otherwise} & \alpha \text{ shifted by } T \text{ to the left} \end{cases}.$$

Reshaping Output. Reshaping is often introduced because the output of a node usually does not conform anymore with the traffic constraint at the input. In our case, we can place a greedy shaper with shaping curve α after the node. The input to the shaper has an arrival curve α^* given by Equation 6.13. The buffer B required at the shaper, given by $v(\alpha^*, \alpha)$ (see Figure 6.4), is then [$\star$][1]

$$B = \begin{cases} \text{if } \frac{b-M}{p-r} \leq T & (b-M) + rT \\ \text{otherwise if } p > R & RT + \frac{(b-M)(p-R)}{p-r} \\ \text{otherwise} & pT \end{cases}. \tag{6.14}$$

[1] [BT12, p. 32] uses $<$ rather than $\leq$ in the first case. This does not make any difference because if $\frac{b-M}{p-R} = T$ the values of the three cases are equal. However, we chose $\leq$ to be consistent with Equation 6.13.
Another difference with [BT12, p. 32] is that we substracted M from the results they have. This is because they consider that $\alpha^*(0) \neq 0$. In this case, the maximum vertical deviation is, for the three cases, achieved in 0 where $\alpha(0) = 0$. However, we have seen that α^* can be replaced by $\alpha^* \wedge \delta_0$ (See footnote 1 on page 23). Hence we can consider $\alpha^*(0) = 0$ and the maximum vertical deviation is then achieved in 0^+. Since we have $\alpha(0^+) = M$, substracting M from the results in [BT12, p. 32] still gives better yet still valid bounds.

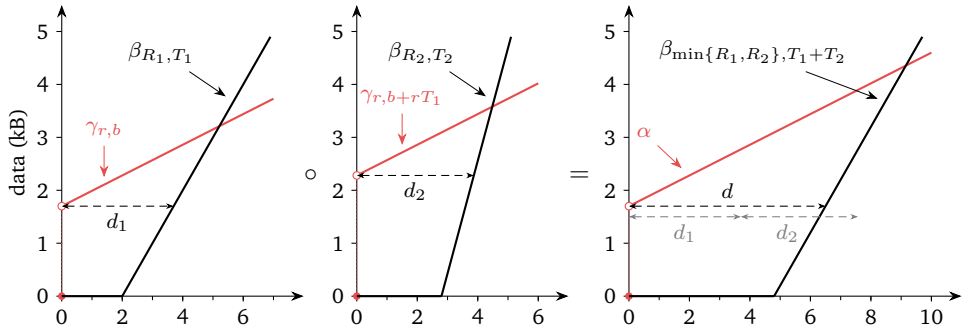


Figure 6.5: The delay bound computed on the convolved system is lower than the sum of the bounds computed on the individual systems (PBOO principle).

6.3 Pay Bursts Only Once (PBOO)

The concatenation theorem allows us to derive an important network calculus result: the *pay bursts only once* (PBOO) principle [BT12, pp. 28-29].

Assume a $\gamma_{r,b}$-smooth flow goes through a series of two rate-latency servers β_{R_1,T_1} and β_{R_2,T_2}. From the example of Section 6.2.1, we know that the delay experienced at the first node is bounded by $d_1 = T_1 + b/R_1$. The flow is then $\gamma_{r,b+rT_1}$-smooth when entering the second system. The delay experienced at the second node is hence bounded by $d_2 = T_2 + (b + rT_1)/R_2$. All in all, the total delay is bounded by

$$d = d_1 + d_2 = T_1 + T_2 + \frac{b}{R_1} + \frac{b}{R_2} + \frac{rT_1}{R_2}. \tag{6.15}$$

If we now compute the delay bound by considering the whole system, i.e. by using the convolution of the two service curves, we obtain

$$d = T_1 + T_2 + \frac{b}{\min\{R_1, R_2\}}. \tag{6.16}$$

We see that the result of Equation 6.16 is always smaller than 6.15. This is shown in Figure 6.5.

Elements of the form b/R_i represent the delay due to the burstiness of the input flow. We say that we *pay bursts only once* because Equation 6.15 contains two such elements while Equation 6.16 has only one.

A similar observation can be made for the backlog bound. These observations are the essence of the PBOO principle: *the bounds obtained by considering the service curve of the whole system are always better than the bounds obtained by considering each node in isolation*.

The insight behind this principle is that we know the total worst-case delay and backlog experienced by the flow when traversing the system, but we do not know the contribution of each node to these values. A bound of the contribution of each node

can be obtained by considering each one of them individually. This is for example necessary to compute the buffer requirements at each node. We have to do so because, though we know that the total backlog in the system will be lower than the sum of all the buffer space we allocate, we do not have more information on how the buffered data will be shared among the nodes.

A corollary of this is that the end-to-end delay and backlog bounds experienced by a flow does not depend on the order of the elements that the flow traverses [BT12, p. 29].

6.4 Greedy Shapers Come For Free

If we introduce a greedy shaper on a path, some bits may be delayed at the shaper. Indeed, we know that, if the shaping curve σ of the shaper is sub-additive, the shaper offers its shaping curve as service curve (see Section 5.5). Therefore, a bound on the delay experienced by a flow at the shaper is given by the horizontal deviation between the arrival curve of the flow and the shaping curve of the shaper [Geo+96, p. 484]. One might then think that the overall end-to-end delay will increase. However, this is not always true. Indeed, assume an α-smooth flow traverses two systems $\mathcal{S}_1$ and $\mathcal{S}_2$ in sequence. If a greedy shaper with curve $\sigma \geq \alpha$ is inserted between $\mathcal{S}_1$ and $\mathcal{S}_2$, then the backlog and delay bounds for the system without shaper are also valid for the system with shaper. This phenomenon is known as *greedy shapers come for free* [BT12, pp. 33-34].

Note however that this does not mean that the shaper does not need any buffer. Only the *total* buffer requirement is not increased [BT12, p. 34].

In contrast, adding a greedy shaper has an obvious benefit. The burstiness of the flow admitted at the next node is reduced, and the delay and backlog bounds at this node are hence also lowered.

CHAPTER 7

PACKET-BASED SYSTEMS

7.1 Introduction

The developments so far considered continuous data flows. However, real packet switching systems send data on a per-packet basis (entire packets) rather than bit-by-bit. In this Chapter, we assume that we observe only entire packets [BT12, p. 40].

We know that greedy shapers keep arrival constraints. Let us consider a flow constrained by $\sigma(t) = l_{max} + rt$ traversing a link of capacity $c > r$, i.e. a greedy shaper with shaping curve λ_c. Consider that the flow sends a first packet of size l_{max} at $T_1 = 0$ and a second packet of size $l_2 < \frac{r}{c} l_{max}$ at time $T_2 = \frac{l_2}{r}$. The flow is hence σ-smooth. Now considering packetization, the departure time of the first packet is $T_1' = \frac{l_{max}}{c}$. We have $T_1' > T_2$. Hence the packets are sent back-to-back and the departure time of the second packet is $T_2' = \frac{l_{max}+l_2}{c}$. The spacing between both packets is now less than $\frac{l_2}{r}$ and the output is therefore not anymore σ-smooth. This example (from [BT12, p. 41]) shows that considering packetization invalidates the result that greedy shapers keep arrival constraints.

Fortunately, we will show in this Chapter how it is possible to quantify the irregularities introduced by packetization, which will allow us to still use the concepts introduced in the previous Chapters.

7.2 The Packetizer

Let us consider $L(n)$ ($n \in \mathbb{N}$), the wide-sense increasing sequence of cumulative packet lengths. We then define the following building block [BT12, p. 41]

$$\boxed{P^L(x) = \sup_{n \in \mathbb{N}} \{L(n) : \ L(n) \leq x\}}, \qquad \text{(Function } P^L\text{)}$$

which can be alternatively defined by [BT12, p. 41]

$$P^L(x) = L(n) \ \Leftrightarrow \ L(n) \leq x < L(n+1), \tag{7.1}$$

i.e. $P^L(x)$ is the largest cumulative packet length that is entirely contained in x. It can easily be seen that the function is right-continuous [BT12, p. 41].

An *L-packetizer* is defined as the system that transforms $R(t)$ into $P^L(R(t))$ [BT12, p. 41]. A flow R is then said *L-packetized* if $P^L(R(t)) = R(t)\ \forall t$ [BT12, p. 41].

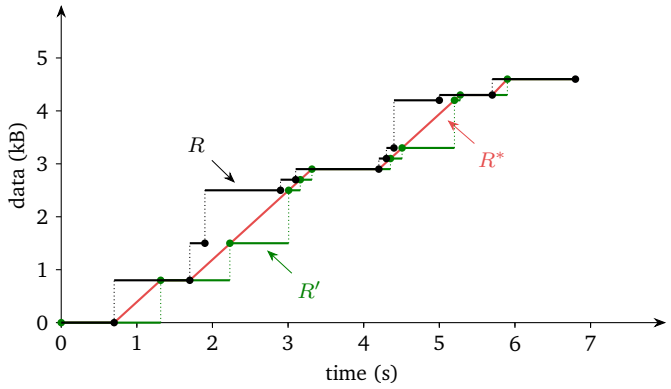


Figure 7.1: Modeling of a real variable length packet trunk with constant bit rate as a greedy shaper (input R, output R^*) followed by a packetizer (input R^*, output R') (adapted from [BT12, p. 40]).

The packetizer enjoys the following properties [BT12, p. 42].

- **Isotonicity.**
  $\forall x, y \in \mathbb{R}, \quad x \leq y \;\Rightarrow\; P^L(x) \leq P^L(y)$
- **Idempotency.**
  $\forall x \in \mathbb{R}, \quad P^L(P^L(x)) = P^L(x)$
- **Optimality.**
  Among all flows such that

$$\begin{cases} x \text{ is L-packetized} \\ x \leq R \end{cases}$$

  $P^L(R(t))$ is the upper-bound.

7.3 Impact of the Packetizer

Figure 7.1 shows an example of L-packetized input flow R along with the corresponding bit-by-bit output R and L-packetized output R' if the flow traverses a greedy shaper with shaping curve λ_c. This corresponds to the correct modeling of a real variable length packet trunk with constant bit rate. We see that the delay and backlog bounds are bigger than those that we would have obtained considering only R and R^*. The following quantifies this deviation.

Consider a bit-by-bit system with L-packetized input R and bit-by-bit output R^* with service curve β and maximum service curve γ. The output R^* is then L-packetized to produce the final output R'. If the systems are FIFO and lossless, we have the following results[1] [BT12, pp. 42-44].

- The per-packet delay[2] for the combined system is the maximum virtual delay for the bit-by-bit system.

[1] $l_{max} = \sup_n \{L(n+1) - L(n)\}$.

[2] For a system with L-packetized input and output, the per-packet delay is $\sup_i \{T'_i - T_i\}$ where T'_i and T_i are the arrival and departure times for the ith packet.

- The service curve β' (from which the maximum backlog for the combined system can be computed) and maximum service curve γ' of the combined system are given by

$$\beta'(t) = [\beta(t) - l_{max}]^+, \tag{7.2}$$

$$\gamma'(t) = \gamma(t). \tag{7.3}$$

- If a flow S has $\alpha(t)$ as an arrival curve, then $P^L(S(t))$ has $\alpha(t) + l_{max}1_{\{t>0\}}$ as an arrival curve.

The second point is consistent with Figure 7.1 while the third one is consistent with the observation made in Section 7.1. The first point can be interpreted as follows. The packetizer waits for the last bit of a packet to consider the first bits transmitted. Therefore, the packet itself is not delayed, since it is fully received at the same time. However, downstream nodes will have to wait for the entire packet to be received before being able to process it. The processing of the packet is then delayed. Packetizers hence do not increase the maximum delay at the node where they are appended but they, however, generally increase the end-to-end delay [BT12, p. 45].

That is why, for end-to-end delay bound calculations, the packetizer at the last hop can be neglected. Consider for example [BT12, p. 44] the concatenation of m GPS nodes with rate R, each node being followed by an L-packetizer. Each GPS node followed by its associated L-packetizer offers (from the result hereabove) a service curve $\beta_{R,\frac{l_{max}}{R}}$. The complete system hence offers a service curve $\beta_{R,m\frac{l_{max}}{R}}$. However, to compute the end-to-end delay bound, we can neglect the last packetizer and consider the service curve $\beta_{R,(m-1)\frac{l_{max}}{R}}$. If the flow is originally constrained by $\gamma_{r,b}$, the end-to-end delay bound is hence

$$\frac{b + (m-1)l_{max}}{R}.$$

7.4 Packetized Greedy Shaper

Consider an L-packetized input sequence $R(t)$.

Packetized Shaper A *packetized shaper* with shaping curve σ is a system that forces its output to have σ as an arrival curve and to be L-packetized [BT12, p. 48].

Packetized Greedy Shaper A *packetized greedy shaper* with shaping curve σ is a packetized shaper that delays the input bits in a buffer whenever sending them would violate the constraint σ, but outputs them as soon as possible [BT12, p. 48]. For a packet to be transmitted, we must of course have $\sigma(t) \geq l_{max} \ \ \forall t > 0$ [BT12, p. 49].

Consider a sequence L of cumulative packets lengths and a good function σ such that there is a sub-additive function σ_0 and a $l \geq l_{max}$ such that

$$\sigma(t) = \sigma_0(t) + l1_{\{t>0\}}.$$

If we call $\mathcal{C}_\sigma$ the greedy shaper with shaping curve σ and $\mathcal{P}_L$ the L-packetizer, for any input, the output of the composition

$$\mathcal{P}_L \circ \mathcal{C}_\sigma \circ \mathcal{P}_L$$

is σ-smooth [BT12, p. 46]. For an L-packetized input, the packetized greedy shaper can be realized as[3] $\mathcal{C}_\sigma \circ \mathcal{P}_L$ [BT12, p. 49]. The assumption on σ is satisfied if σ is concave and $\lim_{t\to 0^+} \sigma(t) \geq l_{max}$ [BT12, p. 46]. If the arrival curve α of an L-packetized flow also satisfies this condition, the output of the L-packetized greedy shaper with shaping curve σ is still α-smooth [BT12, pp. 51-52].

If σ does not satisfy the above condition, the packetized greedy shaper cannot be realized simply as $\mathcal{C}_\sigma \circ \mathcal{P}_L$. If σ is only a good function, the output $\bar{R}$ of the greedy shaper is given by

$$\bar{R} = \inf\{R^{(1)}, R^{(2)}, R^{(3)}, \ldots\}, \tag{7.4}$$

where $R^{(1)} = P^L((\sigma \otimes R))(t))$ and $R^{(i)} = P^L((\sigma \otimes R^{(i-1)}))(t))\ \forall i > 1$ [BT12, p. 50]. $\bar{R}$ is actually the biggest flow smaller or equal to R, L-packetized and σ-smooth [BT12, pp. 50-51].

It can be shown that, for a series of M packetized greedy shapers with shaping curves σ^m such that $\lim_{t\to 0^+} \sigma^m \geq l_{max}$ and for L-packetized inputs, the series is equivalent to a packetized greedy shaper with shaping curve $\sigma = \min_m \sigma^m$ [BT12, pp. 52-53].

[3] We consider that $\mathcal{S}_1 \circ \mathcal{S}_2$ means that $\mathcal{S}_1$ is applied first, which is different from the notation used in [BT12].

CHAPTER 8

SERVICE CURVE DETERMINATION

In the previous Chapters, we have seen what network calculus allows us to compute based on arrival and service curves. Most flows can easily be modeled by an affine arrival curve. We now need to know how to obtain the service curve corresponding to a physical node. This issue is addressed in this Chapter. We will then be able to model a flow with its arrival curve, a node (and a network) with its service curve, and hence to compute bounds in real scenarios.

8.1 Constant Delay Line

A *constant delay line* is a network element that outputs all data which arrives on its single input on its single output stream exactly T seconds later [Cru91, p. 117]. This corresponds to a node with service curve and maximum service curve given by δ_T.

Obviously, the maximum delay experienced at this node is given by T. The output arrival curve α^* is equal to the initial arrival curve α [Cru91, p. 117]. Indeed, $(\alpha \otimes \delta_T) \oslash \delta_T = \alpha$. Besides, the backlog is bounded by $\alpha(T)$. This is shown in Figure 8.1.

This type of element can be used to model propagation and processing delays [Cru91, p. 117]. As these elements do not modify the arrival curve of flows, they can usually be omitted in the modeling. The end-to-end delay bound of a flow can then be obtained by computing the delay bound without these elements and adding their delay contribution to the result.

Since $\delta_T - l = \delta_T \ \forall l$, this model is valid both for bit-by-bit and packet-by-packet systems [⋆].

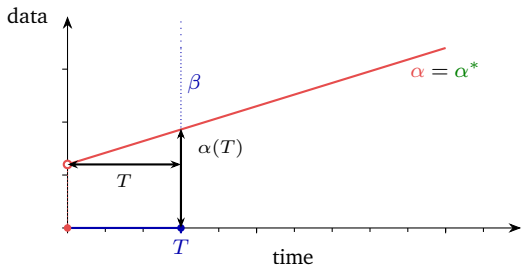


Figure 8.1: Bounds at a constant delay line network element.

8.2 Schedulers

In order to guarantee different service curves to flows, network nodes implement some form of *packet scheduling* (also called *service discipline*) among different flows. Packet scheduling consists in deciding, at the output of a node, the service order for different packets [BT12, p. 67].

8.2.1 First-In First-Out (FIFO)

The simplest form of packet scheduling is *first-in first-out* (FIFO). Packets are served in order of arrival. The different flows are not isolated. If the node is serving the flows at a rate r, it is a greedy shaper with shaping curve $\sigma = rt$. It then offers a service curve $\beta = rt$ to the aggregate flow. The virtual delay and buffer bounds are the same for all the flows and must be computed with the arrival curve of the aggregate flow [BT12, pp. 67-68].

The virtual delay computed in such a way does not correspond to the real delay because a FIFO node is not necessarily FIFO per bit. Indeed, since packets are considered only when fully received, a bit belonging to a small packet might be sent before a bit arrived earlier but belonging to a big packet. For the virtual delay bound to correspond to the per-packet delay bound, we must have an L-packetized input [$\star$]. Indeed, in such a case, bits belonging to the same packet are seen all at the same time and both packets and bits are hence transmitted FIFO. Then, to reflect packetization at the output, the service curve has to be changed to $\beta = [rt - l_{max}]^+$.

8.2.2 Priority Queuing (PQ)

FIFO does not provide any isolation among flows. Under *priority queuing* (PQ) packets arriving at the output link are classified into priority classes. Typically, each class has its own queue and when a packet is to be transmitted, a packet from the highest priority class with the non-empty queue is chosen (choice among packets in the same priority queue is generally done FIFO). Under *nonpreemptive* PQ scheduling, the transmission of a packet is not interrupted once it has begun. In constrast, a *preemptive* PQ scheduler would stop the transmission of a packet if a packet of higher priority arrives at the node (so that the latter can be sent immediately) [KR13, pp. 642-643].

Consider a non preemptive priority scheduler serving two flows H and L. The node offers a *strict* service curve β to the aggregate of the two flows. H has priority over L. If H is α_H-smooth and l^L_{max} is the maximum packet size of L, the service curves guaranteed to H and L are [BT12, p. 176]

$$\boxed{\beta_H = [\beta - l^L_{max}]^+} \qquad \text{(PQ - High Priority)}$$

and

$$\boxed{\beta_L = [\beta - \alpha_H]^+} \qquad \text{(PQ - Low Priority)}$$

if these are wide-sense increasing. β_H is also a strict service curve [BJT09, pp. 19-20]. β_L is not strict but $\beta'_L = [\beta - \alpha_H - l^L_{max}]^+$ is [BJT09, pp. 19-20].

For example, if the node (also called *server*) serves the aggregate at a rate C and if H is $\gamma_{r,b}$-smooth ($r < C$), then [BT12, p. 21]

$$\beta_H = \beta_{C, \frac{l^L_{max}}{C}}, \tag{8.1}$$

$$\beta_L = \beta_{C-r, \frac{b}{C-r}}. \tag{8.2}$$

The latency $\frac{l^L_{max}}{C}$ of β_H accounts for the fact that, since the scheduler is non preemptive, H might have to wait for a complete L packet to be sent. The latency $\frac{b}{C-r}$ of β_L is the time needed to empty the buffer of the high priority queue. Indeed, the buffer might be instantly containing b and then filled at rate r. If emptied at a rate C, the buffer will finally be empty at time $t^* : \ b + rt^* - Ct^* = 0$, i.e. at time $\frac{b}{C-r}$. After this time, L can be served at a rate $C - r$ since r is still used to serve H. In this example, both service curves are strict [BT12, p. 22].

Generalizing to n classes, if C is the overall capacity of the server and α_i the arrival curve for class i (class 1 is the highest priority) then the service curve for class i is [Sch+03, p. 4170]

$$\beta_i(t) = \left(Ct - \sum_{j=1}^{i-1} \alpha_j(t) - \max_{i+1 \leq j \leq n} \{l^j_{max}\} \right)^+. \tag{8.3}$$

From the results above, taking the maximum over $i \leq j \leq n$ makes it strict [BJT09, p. 20].

In the particular case where $\alpha_i = \gamma_{r_i,b_i}$ (i.e. classes are token bucket flows) the service curve for class i is given by [Sch+03, p. 4171] $\beta_i = \beta_{R_i,T_i}$ where

$$R_i = C - \sum_{j=1}^{i-1} r_j, \tag{8.4}$$

$$T_i = \frac{\sum_{j=1}^{i-1} b_j + \max_{i+1 \leq j \leq n} \{l^j_{max}\}}{C - \sum_{j=1}^{i-1} r_j}. \tag{8.5}$$

From the results of Section 6.1, the delay and backlog experienced by class i are then bounded by [Sch+03, pp. 4171-4172]-[Cru91, pp. 122-123]

$$d_i(t) \leq d_i = \frac{\sum_{j=1}^{i} b_j + \max_{i+1 \leq j \leq n} \{l^j_{max}\}}{C - \sum_{j=1}^{i-1} r_j}, \tag{8.6}$$

$$x_i(t) \leq x_i = b_i + r_i \left(\frac{\sum_{j=1}^{i-1} b_j + \max_{i+1 \leq j \leq n} \{l^j_{max}\}}{C - \sum_{j=1}^{i-1} r_j} \right), \tag{8.7}$$

and the new burst of the class after having traversed the scheduler is given by

$$b^*_i = b_i + x_i \tag{8.8}$$

while its rate is unchanged.

Note that these formulas do not consider the packetization of the output of the scheduler. To do so, each obtained service curve β must be transformed as explained in Section 7.3.

8.2.3 Generalized Processor Sharing (GPS)

The ideal form of per-flow queuing is known as *generalized processor sharing* (GPS). Each flow i going through the flow is allocated a weight ϕ_i. A GPS node with a total output rate of C guarantees to flow i that the rate it will receive is zero if flow i has no backlog and otherwise is

$$\frac{\phi_i}{\sum_{j \in B(t)} \phi_j} C,$$

where $B(t)$ is the set of backlogged flows at time t [BT12, p. 68].

It can be easily seen that a GPS node offers to flow i a service curve $\beta_{\frac{\phi_i C}{\sum_j \phi_j},0}$ [BT12, pp. 18, 68]. Hence, for every flow, if we know its weight and its arrival curve, delay and backlog bounds can be computed. However, GPS is a theoretical concept which is not really implementable because it relies on the fluid model and assumes that packets are infinitely divisible.

8.2.4 Packet by Packet GPS (PGPS)

A practical implementation of GPS could consist in, for every packet, computing its finish time under GPS and then present the packet to the output link at its finish time. However, such a scheduler might not be work-conserving, i.e. might be idle at some times while having some packets waiting to be transmitted [BT12, p. 68].

Another practical implementation of GPS, which is work-conserving, is *packet by packet GPS* (PGPS). For every packet, its finish time under GPS is computed. Then, whenever a packet is finished transmitting, the next packet selected for transmission is the one with the earliest GPS finish time. The work-conserving property comes at the expense of a packet being possibly transmitted before its GPS finish time [BT12, p. 68]. It can then be shown that the finish time of PGPS is at most the finish time of GPS plus L/C where L is the maximum packet size (among all flows present at the scheduler) and C the transmission rate [BT12, p. 68].

The following Section describes a powerful framework allowing to model most of the practical implementations of GPS, including PGPS.

8.2.5 Guaranteed Rate (GR)

8.2.5.1 The Framework

There exist a lot of different implementations of GPS that differ in their implementation complexity and in the bounds that can be obtained. Most of them fit in the *guaranteed rate* framework, which is based on the max-plus algebra [BT12, p. 70]-[GLV95]. The idea is that we now deal with the arrival and departure times of packets, rather than with cumulative functions.

Consider a node serving a flow whose packets are numbered, starting from 1, in order of arrival. If $a_i \geq 0$, $d_i \geq 0$ and l_n are the arrival time, departure time and length of packet i, we say that a node is a GR node with rate r and delay e for this flow if it

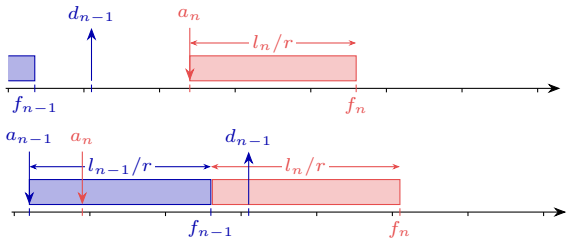


Figure 8.2: Illustration of how a GR node computes the f_i of a packet depending on whether the packet arrives before or after f_{n-1} (adapted from [BT12, pp. 197-198]).

guarantees that[1] [BT12, pp. 70-71]

$$\boxed{\begin{array}{c} d_n \leq f_n + e \\ \begin{cases} f_0 = 0 \\ f_n = \max\{a_n, f_{n-1}\} + \frac{l_n}{r} \quad \forall n \geq 1 \end{cases} \end{array}}, \qquad \text{(GR Node)}$$

or alternatively, that $\forall n$ there is a $k \in \{1, ..., n\}$ such that

$$d_n \leq e + a_k + \frac{l_k + \ldots + l_n}{r}. \qquad (8.9)$$

Figure 8.2 illustrates how the f_i are computed. On the one hand, if packet n arrives after f_{n-1} (upper Figure), then its f_n is computed starting from a_n. On the other hand, if packet n arrives before f_{n-1} (lower Figure), then its f_n is computed starting from f_{n-1}.

For example, an ideal GPS node is a GR node with rate $\frac{\phi_i}{\sum_j \phi_j} C$ and latency 0 [BT12, p. 70].

8.2.5.2 Characterization of Nodes as GR Nodes

A scheduler S is said to *deviate from GPS by* e if, for any packet n, the departure time d_n is such that $d_n \leq g_n + e$ where g_n is the departure time from the theoretical GPS node that allocates a rate r to this flow [BT12, p. 70]. Such a scheduler is then a GR node with rate r and latency e [BT12, p. 70].

For example,

- a PGPS node is a GR node with rate $\frac{\phi_i}{\sum_j \phi_j} C$ and latency $\frac{L}{C}$ [BT12, p. 70],
- a node (not necessarily FIFO) guaranteeing a delay $\leq \delta_{max}$ is, $\forall\, r > 0$, GR with rate r and latency $[\delta_{max} - l_{min}/r]^+$, where l_{min} is the minimum packet size [BT12, p. 72],
- the concatenation (in any order) of a FIFO node imposing a delay $\leq \delta_{max}$ and a FIFO GR node with rate r and latency e is GR with rate r and latency $e + \delta_{max}$ [BT12, p. 73],

[1] Note that a GR node need not be FIFO.

- the concatenation of a first (not necessarily FIFO) node imposing a delay $[\delta_{max} - \delta, \delta_{max}]$ with a FIFO GR node with rate r and latency e is a GR node with rate r and latency $e + \delta_{max} + (\alpha(\delta) - l_{min})/r$ where α is a continuous arrival curve of the fresh input [BT12, p. 73].

If a FIFO node with L-packetized input guarantees a service curve equal to $\beta_{r,e}$, then it is a GR node with rate r and latency e [BT12, p. 71]. A GR node with rate r and latency e is the concatenation of a rate-latency node $\beta_{r,e}$ and an L-packetizer. If the GR node is FIFO, then the service curve element is also FIFO [BT12, p. 71].

From this, a GR node offers a service curve[2]

$$\boxed{\beta = \beta_{r,e+\frac{l_{max}}{r}}}, \qquad \text{(GR Node Service Curve)}$$

which is valid for packet-by-packet systems. Because many implementations of GPS can be quantified in terms of their deviation from GPS, this formula allows to easily derive a service curve for a scheduler emulating GPS.

For an α-smooth flow going through a GR node (not necessarily FIFO), the delay for any packet is bounded by [BT12, p. 71]

$$\sup_{t>0} \left\{ \frac{\alpha(t)}{r} - t \right\} + e, \tag{8.10}$$

i.e. by the horizontal deviation between the arrival and service curve. This result is stronger than the usual delay bound because the GR node need not to be FIFO.

8.2.5.3 Concatenation of GR Nodes

Concatenating GR nodes that are FIFO per flow follow the classical rule of concatenation of service curves and packetizers. This means that the concatenation of M FIFO GR nodes is a GR node with [BT12, p. 72] rate

$$r = \min_m r_m \tag{8.11}$$

and latency

$$e = \sum_{i=1}^{n} e_i + \sum_{i=1}^{n-1} \frac{l_{max}}{r_i}. \tag{8.12}$$

A bound on the end-to-end delay of a $\gamma_{r,b}$-smooth flow through a concatenation of GR nodes is thus [BT12, p. 72]

$$\sum_{m=1}^{M} e_m + l_{max} \sum_{m=1}^{M-1} \frac{1}{r_m} + \frac{b}{\min_m r_m}. \tag{8.13}$$

This result is no longer true for GR nodes that are not FIFO per flow [BT12, p. 73].

[2] That is why a GR node is also called a *rate-latency server*.

8.3 Flows Aggregation

In the previous Section, we have shown how to compute the service curve offered to different flow classes at a scheduler. However, we know that several flows can be classified into the same class. Hence, the service curves computed might be offered to an aggregate of flows. In order to be able to obtain bounds for the individual flows and not only for the aggregate, we will try in this Section to derive the service curve offered to the individual flows of an aggregate. Unfortunately, the state of the art dealing with aggregate multiplexing is not very rich [BT12, p. 175].

Without loss of generality, we will only consider two flows.

8.3.1 Strict Service Curve Element

Consider two flows being served, with some unknown arbitration between the two flows, by a node guaranteeing a *strict* service curve β. If flow 2 is α_2-smooth, then, if it is wide-sense increasing,

$$\boxed{\beta_1 = [\beta - \alpha_2]^+} \qquad \text{(Residual Service Curve - Strict Service Curve Node)}$$

is a service curve for flow 1 [BT12, p. 176]-[BJT09, pp. 17-18]. This is when we do not know anything about the scheduling between flows (also called *blind multiplexing*). If flow 2 has priority over flow 1, the service curve is strict [BJT09, pp. 17-19]. This shows that there is a difference between blind multiplexing and fixed priorities, though the worst departure process for blind multiplexing is fixed priorities. [BJT09, p. 19] provides a nice example to show how this is possible.

For example, considering a node with strict service curve $\beta_{R,T}$ serving two leaky bucket flows with parameters (r_i, b_i), we have that [BT12, pp. 176-177]-[Cru91, pp. 121-122][3], if $r_1 + r_2 \leq R$, the output of flow 1 is a leaky bucket with parameters $r_1^* = r_1$ and

$$b_1^* = b_1 + r_1 T + r_1 \frac{b_2 + r_2 T}{R - r_2}.$$

8.3.2 FIFO Service Curve Element

The result above does not hold if the service curve of the node is not strict because we can then not bound the duration of the busy period for the other (assumed higher priority) flows [BT12, p. 177]. However assuming the node is lossless, FIFO and guarantees β as a service curve to the two flows, if packet arrivals are instantaneous, then $\forall\ \theta \geq 0$ [BT12, pp. 177-178]

$$\boxed{\beta_\theta^1(t) = \begin{cases} [\beta(t) - \alpha_2(t-\theta)]^+ & \text{if } t > \theta \\ 0 & \text{otherwise} \end{cases}} \qquad \text{(Residual Service Curve - FIFO Node)}$$

are such that $R_1' \geq R_1 \otimes \beta_\theta^1$. Hence the β_θ^1 that are wide-sense increasing are service curves for flow 1.

[3] This reference considers $T = 0$ and adds an additional parameter V which is 0 in our case.

From this, we cannot conclude that $\inf_\theta\{\beta_\theta^1\}$ is a service curve [BT12, p. 178]. However, we can show that the output of flow 1 is constrained by [BT12, pp. 178-179]

$$\boxed{\alpha_1^*(t) = \inf_{\theta \geq 0}\{(\alpha_1 \oslash \beta_\theta^1)(t)\}}. \qquad \text{(Output Bound - FIFO Node)}$$

Considering a FIFO node guaranteeing a service curve $\beta_{R,T}$, if flow 1 is constrained by a leaky bucket with parameters (r_1, b_1) and flow 2 by a sub-additive arrival curve α_2, then [BT12, pp. 179-180], if $r_1 + r_2 < R$[4], the output of flow 1 is a leaky bucket with parameters $r_1^* = r_1$ and

$$b_1^* = b_1 + r_1 T + r_1 \frac{\sup_{t \geq 0}\{\alpha_2(t) + r_1 t - Rt\}}{R}. \tag{8.14}$$

In particular, if α_2 is a leaky bucket with parameters (r_2, b_2), then [BT12, p. 180] flow 1 is guaranteed a service curve $\beta_{R',T'}$ with $R' = R - r_2$ and $T' = T + b_2/R$ and the output of flow 1 is a leaky bucket with parameters $r_1^* = r_1$ and

$$b_1^* = b_1 + r_1 T + r_1 \frac{b_2}{R}. \tag{8.15}$$

This is a better bound than in the strict service curve case, at the expense of the FIFO assumption.

8.3.3 GR Node

Let us now consider the case of a GR node (which is not necessarily FIFO, i.e. arbitration between the two flows is unspecified) with rate R and latency T. If flow 1 is constrained by a leaky bucket with parameters (r_1, b_1) and flow 2 by a sub-additive arrival curve α_2, then [BT12, pp. 180-181], if $r_1 + r_2 < R$, the output of flow 1 is a leaky bucket with parameters $r_1^* = r_1$ and

$$b_1^* = b_1 + r_1 T + r_1 \hat{D}, \tag{8.16}$$

where $\hat{D} = \sup_{t>0}\left\{\frac{\alpha_2(t) + r_1 t + b_1}{R} - t\right\}$. Indeed, from the properties of GR nodes, the delay for any packet is bounded by $\hat{D}+T$ and the output of flow 1 has thus $\alpha_1(t+\hat{D}+T)$ as an arrival curve.

In particular, if α_2 is a leaky bucket with parameters (r_2, b_2), then [BT12, p. 181] the output of flow 1 is a leaky bucket with parameters $r_1^* = r_1$ and

$$b_1^* = b_1 + r_1 T + r_1 \frac{b_1 + b_2}{R}. \tag{8.17}$$

This bound is less good than in the FIFO case, but it is valid for any GR node, even not FIFO.

[4] Where $r_2 = \inf_{t>0}\{\alpha_2(t)/t\}$ is the maximum sustainable rate of α_2.

ACKNOWLEDGMENTS

The authors would like to thank Jochen Guck for the countless discussions and their student Sean Rohringer for his useful feedback for improving the understandability of the document.

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 671648 (VirtuWind).

REFERENCES

[AE10] Robert A. Adams and Christopher Essex. *Calculus. A Complete Course*. 7th ed. Pearson Education Canada, 2010.

[BJT09] Anne Bouillard, Laurent Jouhet, and Eric Thierry. “Service Curves in Network Calculus: dos and don’ts”. In: *Research Report 7094, INRIA* (2009).

[BO62] Andrew M. Brucker and E. Ostrow. “Some Function Classes Related to the Class of Convex Functions”. In: *Pacific Journal of Mathematics* 12.4 (Apr. 1962), pp. 1203–1215.

[BT12] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Online Version, Apr. 2012.

[Bac+92] François Baccelli et al. *Synchronization and Linearity, An Algebra for Discrete Event Systems*. John Wiley and Sons, 1992.

[CBL05] Florin Ciucu, Almut Burchard, and Jörg Liebeherr. “A Network Service Curve Approach For the Stochastic Analysis of Networks”. In: *ACM SIGMETRICS performance evaluation review*. Vol. 33. 1. ACM. 2005, pp. 279–290.

[Cha00] Cheng-Shang Chang. *Performance Guarantees in Communication Networks*. Springer Verlag, 2000.

[Cru91] Rene L. Cruz. “A Calculus for Network Delay, Part I: Network Elements in Isolation”. In: *IEEE Transactions on Information Theory* 37.1 (1991), pp. 114–131.

[ELL90] A.E. Eckberg, Daniel T. Luan, and David M. Lucantoni. “An Approach to Controlling Congestion in ATM Networks”. In: *International Journal of Digital & Analog Communication Systems* 3.2 (1990), pp. 199–209.

[Fid06] Markus Fidler. “WlC15-2: A Network Calculus Approach to Probabilistic Quality of Service Analysis of Fading Channels”. In: *Global Telecommunications Conference, 2006. GLOBECOM’06. IEEE*. IEEE. 2006, pp. 1–6.

[GLV95] Pawan Goyal, Simon S. Lam, and Harrick M. Vin. “Determining end-to-end Delay Bounds in Heterogeneous Networks”. In: *Network and Operating Systems Support for Digital Audio and Video*. Springer. 1995, pp. 273–284.

[Geo+96] Leonidas Georgiadis et al. “Efficient Network QoS Provisioning based on per Node Traffic Shaping”. In: *IEEE/ACM Transactions on Networking* 4.4 (1996), pp. 482–501.

[JL08] Yuming Jiang and Yong Liu. *Stochastic Network Calculus*. Vol. 1. Springer, 2008.

[Jia06] Yuming Jiang. “A Basic Stochastic Network Calculus”. In: *ACM SIGCOMM Computer Communication Review* 36.4 (2006), pp. 123–134.

[KR13] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. 6th ed. Pearson Education, 2013.

[MOA11] Albert W. Marshall, Ingram Olkin, and Barry C. Arnold. *Inequalities: Theory of Majorization and Its Applications*. 2nd ed. Springer Series in Statistics, 2011.

[RFC1633] R. Braden, D. Clark, and S. Shenker. *Integrated Services in the Internet Architecture: an Overview*. RFC 1633. RFC Editor, June 1994.

[Ros50] R.A. Rosenbaum. "Sub-additive Functions". In: *Duke Mathematical Journal* 17.3 (1950), pp. 227–247.

[ST83] Ian Stewart and David Tall. *Complex Analysis (The Hitchhiker's Guide to the Plane)*. Cambridge University Press, 1983.

[Sch+03] Jens Schmitt et al. "Per-flow Guarantees under Class-Based Priority Queueing". In: *GLOBECOM '03. IEEE Global Telecommunications Conference* 7 (2003), pp. 4169–4174.

[TW11] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks*. 5th ed. Prentice Hall, 2011.

[Tan03] Andrew S. Tanenbaum. *Computer Networks*. 4th ed. Prentice Hall, 2003.