Scaling, Packetizers and Aggregation in Network Calculus

Damien Guidolin-Pina, Marc Boyer

Combining scaling and packetizers regarding the aggregation of flows in the Network Calculus theory

Damien GUIDOLIN--PINA
RealTime-at-Work
Nancy, FRANCE
damien.guidolin@realtimeatwork.com
https://orcid.org/0000-0003-1149-0861

Marc BOYER
DTIS, ONERA, Université de Toulouse
31000,Toulouse,FRANCE
marc.boyer@onera.fr
https://orcid.org/0000-0003-0344-6991

April 26, 2024

Abstract

Real-time systems often consist of numerous subsystems engaged in extensive data exchange. Despite their complexity, a critical challenge lies in effectively incorporating real-time constraints within these systems. To address this challenge, designers typically conduct analyses to establish upper bounds on delays, ensuring they remain within the deadlines of incoming requests. However, adopting a pessimistic approach often results in over-dimensioning the systems. Then, to reduce the pessimism, we want to take into account the fact that a subsystem cannot propagate more requests/data than it can execute. This phenomenon is well-known in network analysis as it reduces the burst of data. As a consequence, this notion is easier to grasp in theories developed to compute delay bounds in networks. That is why we choose, in this paper, to perform the analysis using the Network Calculus theory, since it offers the possibility to easily aggregate flows (*i.e.* sum flows) and then take into account the phenomenon of smoothing the traffic. To handle tasks *and* networks, our model relies on *packetization* and *workload scaling*. In this paper, we improve some results regarding the already existing elements of Network Calculus and the aggregation. Also, we update and complete definitions and results related to workload scaling.

Keywords— R eal-Time System, Response Time Analysis, Network Calculus, Aggregation of Flows

1 Introduction

Nowadays, real-time systems are increasingly complex and can be composed of a large number of subsystems exchanging a large number of requests/data. One of the challenges is to be able to guarantee that each request will meet its deadline. To do so, several analyses of such real-time systems can allow designers to compute worst-case delay upper bounds and make sure the time constraints are met (if the bounds are lower than the deadlines). However, the analysis can be pessimistic and can lead to oversizing of the system if the bounds are too large wrt. the worst case.

To reduce this pessimism, a little-considered phenomenon could be taken into account: a subsystem cannot propagate more requests/data than it can execute. It means that two propagated requests from the same element of the system are, at least, separated by the time to execute the first one. In a network, it is a phenomenon known as it reduces the bursts of data, *i.e.* the accumulation of data at the same time.

In this paper, we model this phenomenon in distributed systems where tasks communicate through a network and some tasks are released by the reception of a message or the end of the execution of a previous one (released task chains). As a consequence, we will be looking at task chains communicating through a network and computing delay bounds on these chains.

Various theories perform the analysis of such chains. However, as we want to perform bounds on chains communicating through networks like Time Sensitive Networking networks (TSN, [1]) and taking into account the phenomenon described above, we choose to perform the analysis using the Network Calculus theory. One of the strengths of Network Calculus is the possibility to aggregate requests modelled as flows (*i.e.* to sum flows) easily and use the build-in *shaping* to model the fact that the total output of a system is limited by its maximal capacity.

To model release task chains in Network Calculus, we first need to model tasks. Inspired by [2], we give the arrival and departure functions for these tasks based on job characteristics (release time, completion time, and workload). To model task chaining we use existing network elements (putting in sequence a server, a packetizer and a multiplier). A packetizer is a network element that waits for the end of the execution of the request before releasing it instantaneously. A multiplier (called *scaling function* in [3]) is an element of the system that can change the size of a flow.

To be able to benefit from the smoothing/shaping capacity in this context, we extend properties of this packetizer and multiplier in the context of aggregated flows.

The paper is organized as follows. Section 2 presents our model of the release task chain with Network Calculus. A state of the art is presented in Section 3 and notations are introduced in Section 4. Next, Section 5 reminds the Network Calculus basics with new results regarding the aggregation in the packetizer. Then, Section 6 presents an element modifying the workload, the *multiplier*, inspired by the *scaling function* from [3]. Finally, we use the proven results in Section 7 with a sample.

2 System model

Model of tasks and release task chains A task, which we will note $\mathcal{T}$, is an infinite sequence of jobs: $(\mathcal{J}_n)_{n\in\mathbb{N}}$. And a job, $\mathcal{J}$, is characterized by a tuple $\langle r; w; e\rangle$ where r is the release time (*i.e.* the time when the job is ready to start), w the workload (*i.e.* the number of CPU cycles required to execute the job) and $e \geq r$ the completion time (*i.e.* the time when the all workload has been served). We assume that the jobs of a task are ordered in ascending order of the release times.

Regarding the task chains, there are two types, both predicated on a precedence relation between tasks ($\mathcal{T} \to \mathcal{T}'$): "release task chain" where the end of the execution of a job from $\mathcal{T}$ releases a job of $\mathcal{T}'$ and "triggered task chain" where the jobs are independently released but each job of $\mathcal{T}'$ uses the results of the last executed jobs of $\mathcal{T}$. Here, the release time chains are considered, *i.e.* two tasks $\mathcal{T}, \mathcal{T}'$ are chained if the completion time of the job $\mathcal{J}_i$ of the task $\mathcal{T}$ is the release time of the

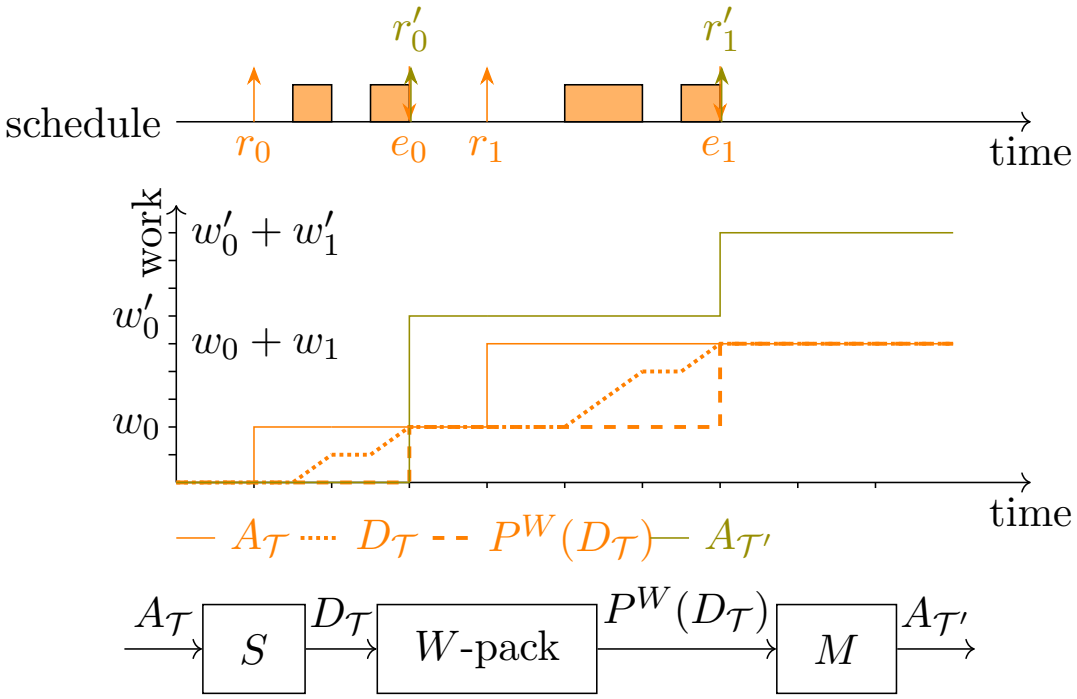

Figure 1: Illustration of the passing from $\mathcal{T}$ to $\mathcal{T}'$, two chained tasks.

job $\mathcal{J}'_i$ of the task $\mathcal{T}'$: $\forall n \in \mathbb{N}, r'_n = e_n$.

Model in Network Calculus Inspired by [2], we propose a way to model the tasks using in Network Calculus. When modelling a network, a cumulative curve represents the cumulative amount of data up to the current instant. When modelling a task $\mathcal{T}$, it represents the amount of workload (number of CPU cycles) generated by a task. Then, it can be defined as

$$A_{\mathcal{T}}(t) = \sum_{n \in \mathbb{N} | r_n \leq t} w_n.$$

Network Calculus models the task scheduling by counting the work done into the cumulative curve $D_{\mathcal{T}}$ (cf. Figure 1).

To model a task chain $\mathcal{T} \to \mathcal{T}'$, we need to derive the arrival $A_{\mathcal{T}'}$ from $A_{\mathcal{T}}$. First, the completion times of jobs can be can be derived from $A_{\mathcal{T}}$ using the network calculus element called *packetizer* (designed to store bits up to having a full packet, which is equivalent as storing work up to full execution). And to generate the workload w'_i of the next job, one just have to multiply the excuted workload w_i by w'_i/w_i. Figure 1 illustrates this sequence of elements that model the task chain.

In this figure, a task $\mathcal{T}$ with two jobs represented ($\mathcal{J}_0 = \langle r_0, w_0, e_0 \rangle$ and $\mathcal{J}_1 = \langle r_1, w_1, e_1 \rangle$) is chained to a second one, $\mathcal{T}'$ ($\mathcal{J}'_0 = \langle r'_0, w'_0, e'_0 \rangle$ and $\mathcal{J}'_1 = \langle r'_1, w'_1, e'_1 \rangle$ with $r'_0 = e_0$ and $r'_1 = e_1$) as shown at the top of the figure. It can be modelled, in Network Calculus, with an input curve $A_{\mathcal{T}}$

which pass through a server S, a packetizer P^L, and a multiplier M as shown at the bottom of the figure. The curve associated with the output of each element is drawn at the middle of the figure.

3 Related work

Curve-based models The Real-time Calculus (RTC) has been developed as a fork of Network Calculus devoted to the analysis of complex real-time systems. An equivalence between the main equations can be found in [4, 5]. To deal with task chain, and especially the change of workload, it uses a dedicated workload curves per task (γ, γ' for tasks τ, τ'), and the composition $\gamma' \circ \gamma^{-1}$ corresponds to our sequence of packetizer and multiplier elements [6, 7]. They do not address the consideration of aggregation of flows.

In Network Calculus these elements have been studied per se and come with several interesting properties (cf. Sections 5.5 and 6.2).

In the Compositional Performance Analysis (CPA), the model is based on the notion of Event Stream, often denoted η, each event being either a task release or a message reception. Modelling the fact that one task $\mathcal{T}$ releases task $\mathcal{T}'$ is done by transforming the event stream of $\mathcal{T}$, often denoted η, into another event stream η', being the event stream of $\mathcal{T}'$. This model is well established, and most work is done on the response time analysis of such chains, as in [8] (focusing on chains where all tasks do not have the same priorities) and [9] (considering also blocking due to sub-calls).

Conversely to Network Calculus, the workload associated with each job is not in the event stream itself, but as a separated parameter, often denoted q, and added when doing the response time analysis. It then does not have to face the problem of the difference in workload size between releasing and released jobs. But when considering an aggregation of several tasks of different sizes, the analysis must consider each individual workload.

Other models to analyse release task chains In [10], the authors consider only periodic tasks using three parameters: the workload, the period and a deadline. Each task can be a subsequence of tasks defined also by a workload, a deadline and they add a priority. Then, a framework is proposed to analyse the schedulability using the notion of busy periods. However, only the periodic task on a fixed priority scheduling is considered. In our manuscript, we try to model the task and task chain with another framework and be as generic as possible regarding the scheduling policy.

In [11], the aim is not simply to compute the response time of a chain, but to take into account the chain to count deadline misses in a weakly-hard real-time system.

Then, in [12], the task chain is formally defined in Definition 3.2 (it is quite similar to our definition except that it is more specific because, for instance, they include a deadline and the execution policy in the definition of the task chain). They also define the task model in Definition 3.1 considering that a task is defined only by a priority and a worst-case execution time where we choose to base our model on [2]. Since then, [12] suggests an analysis of the task chain based on the busy window.

A holistic approach, analysing a system made of different computing resources can also be done [13].

Model cooperation It is shown in [14] how two methods (CPA and RTC) can be combined to analyse a single case study, by analysing each element with one or the other method. One can

also analyse each element with two methods, and get the best of both [15] with CPA and network calculus. The task response time analysis can be used to generate a data flow arrival curve taking into account the task scheduling [16].

Modelling task in Network Calculus A formal equivalence between subset of Network Calculus and the Classical Response Time Analysis presented in the [2]. Our work extends the results to be able to analyse chains of tasks using only the elements of the Network Calculus that are already in place.

4 Notations

We first introduce some notations useful for the following sections.

First, $\mathbb{R}$ and $\mathbb{R}^+$ denote the set of reals and non-negative reals respectively. $\mathbb{N}$ denotes the set of natural integers and $\mathbb{Z}$ the set of integers. $\circ$ represents the composition operator, *i.e.* let f, g be two fonctions from $\mathbb{R}^+$ to $\mathbb{R}^+$ and t a non-negative real, $f \circ g(t) = f(g(t))$.

Also, $\wedge$ denotes the minimum operator *i.e.* $a \wedge b \stackrel{\text{def}}{=} \min(a, b)$.

$\delta_0 : \mathbb{R}^+ \to \overline{\mathbb{R}^+}$ denotes the function such that $\delta_0(0) = 0$ and $\forall t > 0, \delta_0(t) = +\infty$.

Also, we note $U : \mathbb{N} \to \mathbb{R}$ the function associated to a sequence $(U_n)_{n\in\mathbb{N}}$ such that $U(n) = U_n$. Then, we will either note U_n or $U(n)$.

Finally, $\mathcal{F}$ denotes the set of functions from $\mathbb{R}^+$ to $\mathbb{R}^+$ piecewise continuous, $\mathcal{R}$ the subset of $\mathcal{F}$ such that the functions are right continuous. And, for any $X \subseteq \mathcal{F}$, $X^\uparrow$ denotes the subset of X such that the functions are non-decreasing and X_0 denotes the subset of X such that $\forall f \in X_0, f(0) = 0$.

5 Network Calculus basics

Here start the Network Calculus basics inspired from [5], [17]. This section gives the base of the Network Calculus that we need in the following. First, we introduce mathematical basics, then we give definitions and properties regarding the cumulative curves, the arrival curves, and the network elements (servers, packetizers and multipliers).

5.1 Operators

In the Network Calculus theory, the main operators are the (min,plus) convolution (noted $*$), the (min,plus) deconvolution (noted $\oslash$), the (max,plus) convolution (noted $\overline{*}$) and the (max,plus) deconvolution (noted $\overline{\oslash}$), defined respectively, for all $f, g \in \mathcal{F}$ two functions and for all $t \in \mathbb{R}^+$, by

$$(f * g)(t) \stackrel{def}{=} \inf_{0\le v\le t} \{f(t-v) + g(v)\},$$

$$(f \oslash g)(t) \stackrel{def}{=} \sup_{v\ge 0} \{f(t+v) - g(v)\}.$$

$$(f \,\overline{*}\, g)(t) \stackrel{def}{=} \sup_{0\le v\le t} \{f(t-v) + g(v)\},$$

$$(f \overline{\oslash} g)(t) \stackrel{def}{=} \inf_{v \geq 0} \{f(t+v) - g(v)\}.$$

Note that it is possible to move a constant out of the convolution [17, Rule 7 of Theorem 3.1.5], *i.e.* let $f, g \in \mathcal{F}$ and $K \in \mathbb{R}^+$. Then,

$$(f * g) + K = f * (g + K). \tag{1}$$

Also, other operators and definitions are needed in this paper.

Definition 1 (Sub/Super-additivity)**.** *A function $f \in \mathcal{F}$ is said to be super-additive if for all $x, y \in \mathbb{R}^+$, $f(x+y) \geq f(x) + f(y)$.*

The same way, f is said to be sub-additive if for all $x, y \in \mathbb{R}^+$, $f(x+y) \leq f(x) + f(y)$.

Property 1 (Sub/Super-additivity and convolution)**.** *Let $f, g, h \in \mathcal{F}^{\uparrow}$. If h is sub-additive, then $h(f * g) \leq (h \circ f) * (h \circ g)$.*

If h is super-additive and if h is right-continuous or f, g are left-continuous, then $h(f * g) \geq (h \circ f) * (h \circ g)$.

Proof. Let $t \in \mathbb{R}^+$.

By definition of the infimum $\forall s \in \mathbb{R}^+, 0 \leq s \leq t, f * g \leq f(t-s) + g(t)$. As h is non-decreasing, $h(f * g) \leq h(f(t-s) + g(t))$. If h is sub-additive, $h(f(t-s) + g(t)) \leq h(f(t-s)) + h(g(s))$. Then, $h(f * g) \leq h(f(t-s)) + h(g(s))$. As is it right for all s, we have $h(f * g) \leq h \circ f * h \circ g$.

Also, if f and g are left-continuous: we have that $\exists s \in \mathbb{R}^+, 0 \leq s \leq$ such as $f * g(t) = f(s) + g(t-s)$. As h is nondecreasing, $h(f * g)(t) = h(f(s) + g(t-s))$ and as h is super-additive, $h(f * g)(t) \geq h \circ f(s) + h \circ g(t-s) \geq \inf_{0 \leq s \leq t} \{h \circ f(s) + h \circ g(t-s)\}$. Otherwise, if h is right-continuous: by definition of the infimum, $\forall \varepsilon > 0, \exists s_\varepsilon$ such that $f * g(t) + \varepsilon > f(t - s_\varepsilon) + g(s_\varepsilon)$. As h is nondecreasing, we have $h(f * g(t) + \varepsilon) > h(f(t - s_\varepsilon) + g(s_\varepsilon))$. If h is super-additive, $h(f(t-s_\varepsilon)+g(s_\varepsilon)) \geq h \circ f(t-s_\varepsilon) + h \circ g(s_\varepsilon)$. Then, $h(f * g(t) + \varepsilon) \geq \inf_{0 \leq s \leq t} \{h \circ f(t-s) + h \circ g(s)\}$. Using the limit, we then have $h(f * g)(t) \geq \inf_{0 \leq s \leq t} \{h \circ f(t-s) + h \circ g(s)\}$. □

Here, it seems that continuity can be a problem in respecting the hypothesis of Property 1. However, in Network Calculus, it is common to consider all functions as left-continuous or right-continuous and not mix different continuities in the same model. This is enough to meet the assumptions because if we are using left-continuous functions, then f and g are left-continuous, and we consider only right-continuous ones, h is then right-continuous.

In the rest of the paper, we assume that all the functions have the same continuity, either right or left continuous.

5.2 Cumulative curves

In Network Calculus, flows are modelled by cumulative functions $A \in \mathcal{F}_0^{\uparrow}$ such that $A(t)$ counts the total amount of data of a flow passed through some observation point up to time t. Since a flow is a cumulative amount of data, it must be a non-decreasing function. The condition on a finite number of discontinuities in a finite interval ($A \in \mathcal{F}$) is related to the discrete aspect of computer behaviour and simplifies the mathematical part. The condition of null value at 0 ($A \in \mathcal{F}_0$) is related

to the fact that all results in network calculus are based on differences: a quantity $A(t)$ has to be understood as $A(t) - A(0)$.

Commonly, the left continuity is preferred and most of the known results are developed with this assumption. However, due to the fact that we use results from [18], we will deal with the right continuous one, in this paper. However, recent results show that it exists, under conditions, a bridge between the two assumptions [19].

One of the strenghs of the Network Calculus theory is the possibility to aggregate flows, *i.e.* the sum of two cumulative curves remains a cumulative curve.

5.3 Arrival curves

The notion of arrival curves is introduced to bound the cumulative curve on any interval of time and for $A \in \mathcal{F}_0^{\uparrow}$: a cumulative curve, $\overline{\alpha} \in \mathcal{F}^{\uparrow}$, respectively $\underline{\alpha} \in \mathcal{F}^{\uparrow}$, is a maximal, respectively minimal, arrival curve for A, if

$$\forall t, d \in \mathbb{R}^+ : \underline{\alpha}(d) \leq A(t+d) - A(t) \leq \overline{\alpha}(d).$$

From Proposition 5.2 in [5], a property regarding the minimum of arrival curves will be useful.

Property 2 (Minimum between maximal arrival curves)**.** *Let $A \in \mathcal{F}_0^{\uparrow}$ be a cumulative curve, and $\overline{\alpha} \in \mathcal{F}^{\uparrow}$ be a maximal arrival curve for A. Then, if $\overline{\alpha}'$ is another maximal arrival curve of A, $\overline{\alpha} \wedge \overline{\alpha}$ is also a maximal arrival curve for A.*

5.4 Servers (size-invariant)

The common servers in the Network Calculus theory are size invariant elements, *i.e.* the output data size is exactly the same as the input ones. With this assumption, the departure amount is at any time less or equal to the arrival amount. Network elements able to change the size of the processed data will be called "multipliers" and will be defined in Section 6.

A server S is defined as a relation in $\mathcal{R}_0^{\uparrow} \times \mathcal{R}_0^{\uparrow}$, associating to each arrival at least one departure (i.e. left-total[1]). A server is not a function since some non-determinism may associate several departures to a given arrival.

Formally, the definition of a server is as follows.

Definition 2 (Servers)**.** *A server $S \subseteq \mathcal{R}_0^{\uparrow} \times \mathcal{R}_0^{\uparrow}$ is a server that satisfies $\forall (A, D) \in S \implies A \geq D$.*
We denote $A \overset{S}{\rightarrow} D$ for $(A, D) \in S$.

There exist various servers with different properties. Let's recall those useful for this paper, from the literature, as the minplus minimal service, the maximal service, and the shaper.

Definition 3 (Kinds of service)**.** *Let S be a server, and $\beta \in \mathcal{F}^{\uparrow}$, $\beta^M \in \mathcal{F}_0^{\uparrow}$, $\sigma \in \mathcal{F}_0^{\uparrow}$.*
The server S offers a min-plus minimal service curve β if $\forall A, \forall D : A \overset{S}{\rightarrow} D \implies D \geq A * \beta$.
It offers a maximal service curve β^M if $\forall (A, D) \in S \implies D \leq A * \beta^M$.
It is a σ-shaper (also said offers a shaping service curve σ) if $\forall (A, D) \in S \implies D \leq D * \sigma$.

[1] A relation S is left-total when $\forall A \in \mathcal{R}_0^{\uparrow}, \exists D \in \mathcal{R}_0^{\uparrow} \mid (A, D) \in S$

Remark 1. *If the input curve A is an aggregate flow,* i.e. *a sum of cumulative curves, and we want to compute the delay of a specific curve, it is necessary to find the service curve associated with this input curve. This service curve is called the residual service curve,* i.e. *the service remaining for the flow of interest, assuming that the other flows are served correctly.*

Now, we can compute an output arrival curve of a server knowing the input arrival curve with Theorem 5.3 from [18].

Theorem 1 (Output arrival curve)**.** *Let S be a server such that S and A be an arrival cumulative curve that has maximal arrival curve $\overline{\alpha}$ and minimal arrival curve $\underline{\alpha}$. Then for all $D \in \mathcal{R}_0^{\uparrow}$ such that $(A, D) \in S$, D has maximal arrival curve $\overline{\alpha}'$ and minimal arrival curve $\underline{\alpha}'$ with*

$$\overline{\alpha}' = \left(\left(\overline{\alpha} * \beta^M\right) \oslash \beta_m\right) \wedge \sigma$$
$$\underline{\alpha}' = \underline{\alpha} * \left(\beta_m \,\overline{\oslash}\, \beta^M\right).$$

Then, let introduce the delay of a server.

Definition 4 (Delay)**.** *Let S be a server, and consider A and D respective cumulative arrival and departure functions: $A \overset{S}{\to} D$.*

For $t \geq 0$, $hDev(A, D, t)$ is the virtual delay associated to flow A at time t:

$$hDev(A, D, t) = \inf\left\{d \in \mathbb{R}^+ \mid A(t) \leq D(t+d)\right\}.$$

Then, the worst-case delay is $hDev(A, D) = \sup_{t \in \mathbb{R}^+} \{hDev(A, D, t)\}$.

Then, we can see a server as a jitter.

Theorem 2 (Server as a jitter [5, Thm. 6.2])**.** *Let S be a server, and A one arrival cumulative process. If the delay in the server for every bit of data always between d_m and d^M, then S offers to A a min-plus service δ_{d^M} and a maximal service curve δ_{d_m}.*

Using the approximation curve, we can compute a bound on the worst-case delay.

Property 3 (Bound on the worst-case delay [5, Thm 5.2])**.** *Let S be a server, $\alpha \in \mathcal{F}^{\uparrow}$, and $\beta \in \mathcal{F}_0^{\uparrow}$. If $(A, D) \in S$ such that A has maximal arrival curve α and S offers a min-plus minimal service curve β, then the maximum delay can be bounded by $hDev(A, D) \leq hDev(\alpha, \beta)$.*

But, if a flow is crossing several servers, we have the pay burst only once phenomenon which is the fact that it is possible to concatenate the servers to compute a smaller delay bound than the one computed with the sum of the delay bounds for each server. This phenomenon is first introduced in [20, Thm. 3] and reformulate using the (min, plus) convolution in [21, Thm. B] but for the following part, we choose the modern formulation given in [17, Thm. 1.4.6] or [5, Thm. 6.1].

Theorem 3 (Concatenation of servers)**.** *A system made of a sequence of two servers, $A \overset{S}{\to} B \overset{S'}{\to} C$, can be seen as a single server $A \overset{S;S'}{\longrightarrow} C$. Moreover, if S, S' respectively offer min-plus service curves β and β', then $S; S'$ is offering a min-plus service curve $\beta * \beta'$.*

5.5 Packetizer

A packetizer is also a size-invariant element which stores bits up to the end of a packet and delivers instantaneously the whole packet, considering that the flow is composed of a sequence of packets. We can, then, consider the packet length sequence as the function that associates the number of packets to its cumulative amount of data.

Definition 5 (Cumulative packet length)**.** *A function $L : \mathbb{N} \to \mathbb{R}^+$ is a cumulative packet length function if*

1. *it is increasing ($n < n' \implies L(n) < L(n')$)*
2. *it is null at 0 ($L(0) = 0$)*
3. *it is divergent ($\lim_{n\to+\infty} L(n) = +\infty$).*

If the i-th packet has size l_i, then, $L(n) = \sum_{i=1}^{n} l_i$, and conversely, $l_i = L(i) - L(i-1)$. The maximal packet size is $l^u = \sup_{n\in\mathbb{N}} \{L(n+1) - L(n)\}$.

Definition 6 (Packetizer, from [17])**.** *Consider a sequence L of cumulative packet lengths. An L-packetizer is the system that transforms the input $A(t)$ into $P^L(A(t))$ with*

$$P^L(x) = L(n) \Leftrightarrow L(n) \leq x < L(n+1).$$

Remark 2. *It comes: $x - l^u \leq P^L(x) < x$.*

Useful properties/definitions regarding the packetizer are given in [5] and [17]. Let us remind them.

Definition 7 (L-packetized, from [17])**.** *We say that a flow $A(t)$ is L-packetized if $\forall t, A(t) = P^L(A(t))$.*

Corollary 1 (Packetizer as a delay , from Corollary 8.2 [5])**.** *Let S be a server and P be a packetizer. If a cumulative arrival function A is L-packetized, then the system $S; P$ offers the pure-delay min-plus service curve $\delta_{hDev(A,S)}$ to A.*

As we want to challenge our model with aggregations, *i.e.* the sum of different cumulative curves (that is one reason why we chose the Network Calculus theory), we may use a packetizer on aggregate flows. However, it is not proved yet. Then let introduce a new property on the arrival curve of the aggregation from a packetizer.

Property 4 (Server/Packetizer on aggregate flows)**.** *Let A_1, A_2 two cumulative curves with respectively α_1, α_2 their arrival curve and $\alpha_{1,2}$ the arrival curve of the aggregate flow. Let S be a server such that $(A_1, A_2) \overset{S}{\to} (B_1, B_2)$. Let P_1^L, P_2^L be two packetizers with maximum packet size l_1^u, respectively l_2^u such that $B_i \overset{P_i^L}{\to} C_i$. If A_1 is L_1-packetized, A_2 is L_2-packetized and S a server with a minplus minimal service curve β_m, a maximal service curve β^M and a shaping curve σ. Let d_1,*

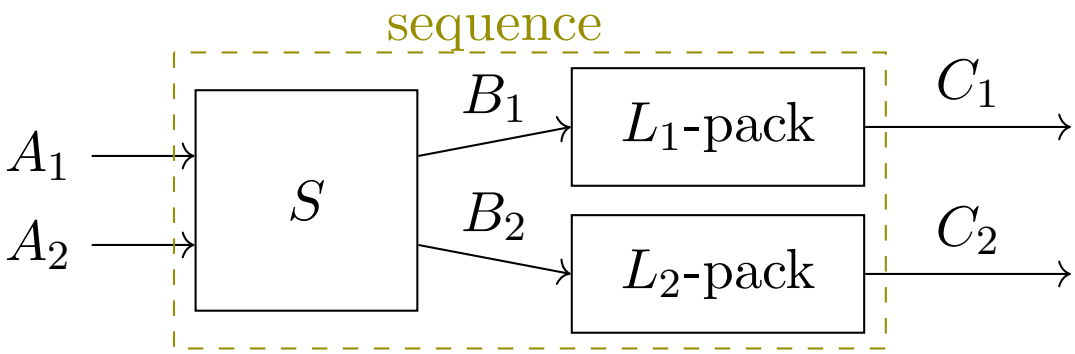

Figure 2: Illustration of two flows passing through a common server and their own packetizer.

respectively d_2, be an upper bound on the delay of the flow A_1, respectively A_2, in the server S. Then, the aggregate cumulative departure function C has maximal arrival curve

$$\min \left(\begin{array}{l} \left(\alpha_{1,2} * \beta^M\right) \oslash \left(\beta_m - (l_1^u + l_2^u)\right) \\ \sigma + (l_1^u + l_2^u) \\ \alpha_{1,2} \oslash \delta_{\max(d_1,d_2)} \end{array} \right)$$

Remark 3. *d_1 and d_2 can be computed with the residual service curve depending to the policy of the server (*e.g. *Static Priority, Round Robin, etc...).*

Figure 2 illustrates the purpose of Property 4. The idea is to have an arrival output curve of the aggregate $C_1 + C_2$ knowing that C_1 and C_2 cross the same server S but are packetized with their own packetizer.

It may be surprising to find $l_1^u + l_2^u$, but it comes from the fact that the flows can be preempted. Indeed, a maximal size packet of A_2 can start to be served, preempted just before the end by a maximal size packet of A_1 and finished to be served after the packet of A_1 is served. As a consequence, the output curve can have a jump of $l_1^u + l_2^u$ bits without any chance to distinguish the packet from A_1 to one of A_2.

Proof. Let note $\hat{S}$ the server such that $A_1 + A_2 \overset{\hat{S}}{\to} C_1 + C_2$.

The proof is composed of two parts:

- prove that the server $\hat{S}$ offers a minimum service curve $\beta_m - (l_1^u + l_2^u)$, a maximal service β^M and a shaping curve $\sigma + (l_1^u + l_2^u)$.

  Then, Theorem 1 gives us the two first terms.

- Prove that the last term is also a maximal arrival curve of C.

Finally, Proposition 2 concludes: the minimum of two arrival curves for C is also an arrival curve for C.

Let $i \in \{1; 2\}$ be an index.

- Let prove the different kinds of curve of $\hat{S}$
  - Minplus minimum service curve: We have a packetizer between B_i and C_i then, according to Remark 2, $C_i \geq B_i - l_i^u$ and we have a minplus minimal service between $A_1 + A_2$ and $B_1 + B_2$, so $(B_1 + B_2) \geq (A_1 + A_2) * \beta_m$. Consequently,

$$\begin{aligned} C_1 + C_2 &\geq B_1 - l_1^u + B_2 - l_2^u \\ &= (B_1 + B_2) - (l_1^u + l_2^u) \\ &\geq (A_1 + A_2) * \beta_m - (l_1^u + l_2^u) \\ &= (A_1 + A_2) * (\beta_m - (l_1^u + l_2^u)) \\ &\text{from Equation. (1)} \end{aligned}$$

    Then, $\beta_m - (l_1^u + l_2^u)$ is a minimal service curve for $\hat{S}$.
  - Maximal service curve: We have a maximal service between A_i and B_i and a server between B_i and C_i. Consequently, $C_1 + C_2 \leq B_1 + B_2 \leq (A_1 + A_2) * \beta^m$. Then, β^m is a maximal service curve for $\hat{S}$.
  - Shaping curve: We have a shaping service between A_i and B_i and we have a server between B_i and C_i. Then,

$$\begin{aligned} C_1 + C_2 \leq B_1 + B_2 &\leq (B_1 + B_2) * \sigma \\ &\leq (C_1 + l_1^u + C_2 + l_2^u) * \sigma \\ &= (C_1 + C_2) * (\sigma + l_2^u + l_1^u) \end{aligned}$$

    Then, $\sigma + l_2^u + l_1^u$ is a shaping curve for $\hat{S}$.

- Let prove the last term is a maximal arrival curve for the aggregate flow. Let d_1, respectively d_2 be an upper bound on the delay of the flow A_1, respectively A_2 passing through S. Using Corollary 1 and Theorem 2, we have $C_1 \leq A_1 * \delta_{d_1}$ and $C_2 \leq A_2 * \delta_{d_2}$. So

$$\begin{aligned} C_1 &\leq A_1 * \delta_{\max(d_1,d_2)} \\ C_2 &\leq A_2 * \delta_{\max(d_1,d_2)} \end{aligned}$$

  As $A * \delta_{\max(d_1,d_2)} = A(t + \max(d_1, d_2))$, then

$$C_1 + C_2 \leq (A_1 + A_2) * \delta_{\max(d_1,d_2)}$$

  Then, the server $\hat{S}$ offers a minplus service curve $\delta_{\max(d_1,d_2)}$. Consequently, $\alpha_{1,2} \oslash \delta_{\max(d_1,d_2)}$ is a valid arrival curve for the aggregate flow.

□

6 Non-size invariant elements

As mentionned Section 5.4, the servers in the Network Calculus theory are size invariant elements, *i.e.* the output data size are exactly the same as the input ones.

This section will introduce multipliers, a renaming of scaling servers. But before giving formal definition, Section 6.1 justify some choices, and in particular that fact that a multiplier has no delay.

6.1 Model justification

They are several flow "transformers" in the network calculus literature: servers (with different flavours: minplus, strict...), packetizers, Pi-regulator [22], and scaling elements [3].

We propose to split into two groups, as illustrated in Figure 5.

To justify this hierarchy and the property of the delay, let us start with an example. Consider the four functions A, B, C, and D plotted in Figure 3.

Function A represents a data flow sending bits at a constant rate from time 0 to 2, then sending nothing from time 2 to time 4, then it re-sends bits, with a given slope during 1 time unit, and slower up to time 7, and then stops.

Function B represents a data flow that sends bits in a constant way, starting at time 1, with different slopes up to time 4. Nothing is sent between 4 and 5, then it restarts at a constant rate up to 7 and then stops.

Function C represents a data flow that sends instantaneously 10 data units at time 5, 20 at time 6 and 15 at time 9. It can represent the emission of three packets, of respective size 10, 20 and 15, but also 9 packets, all of size 5, sending by a burst of 2, 4 and 3 packets.

Similarly, function D represents a data flow that sends instantaneously 16, 32 and 21 units of data at instants 7, 8 and 11.

Consider a sequence of three network elements S, S', S'' such that A, B, C, D are respective input/output (denoted $A \xrightarrow{S} B \xrightarrow{S'} C \xrightarrow{S''} D$).

Note that S and S' matches the current definition of a server, but S'' does not since the rules $D \leq C$ is violated.

The *delay* experienced by a flow crossing a server is defined as the horizontal deviation, denoted $hDev$, between the arrival and the departure curve, which is the smaller deviation such that the arrival goes beyond the departure. The worst delay is of course the supremum of the delay on the set of all possible departures. This definition assumes that all packets inside a flow are served each one after the previous one, what is called 'per-flow FIFO' behaviour[2].

Then, S can model a server receiving a fluid amount of data, and forwarding it with a different fluid shape after a non-constant delay. And S' can model a server storing bit and delivering packets, 2 times units after having received the last bit of the packet.

We may like to interpret S'' has a server encapsulating some packet, forwarding at time 7 a packet received at time 5, and adding an overhead of 6 data units, forwarding at time 8 a packet

[2]If it is not a 'per-flow FIFO' behaviour, we can decompose into several FIFO flows and the Network Calculus also models non-FIFO schedulers such as Round Robin, Static Priority, etc...

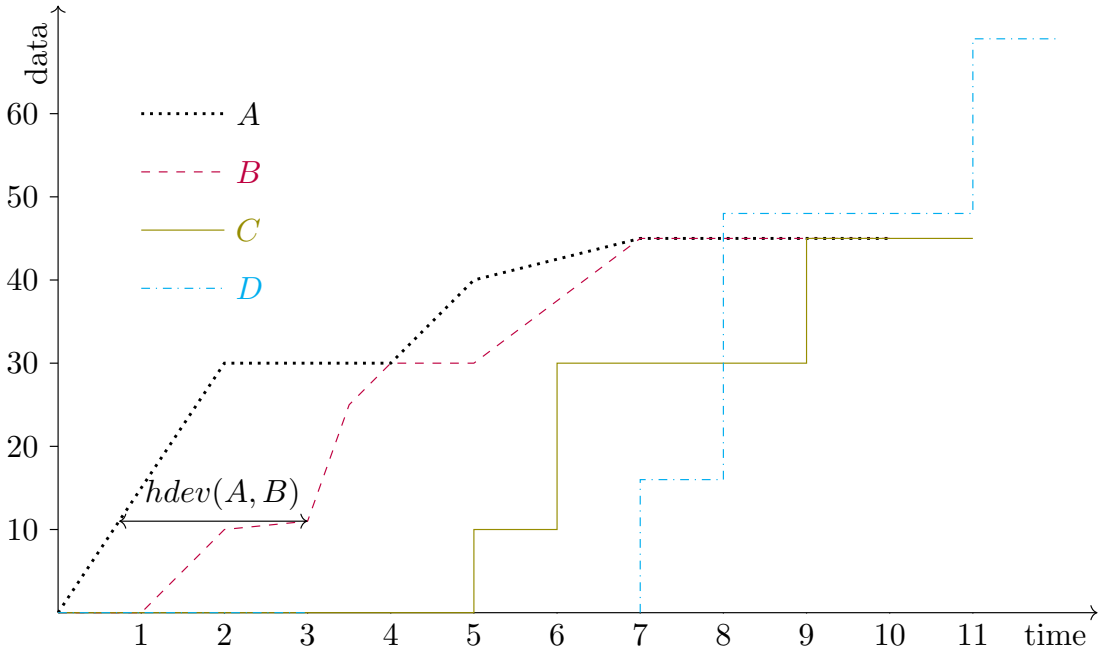

Figure 3: Four cumulative curves.

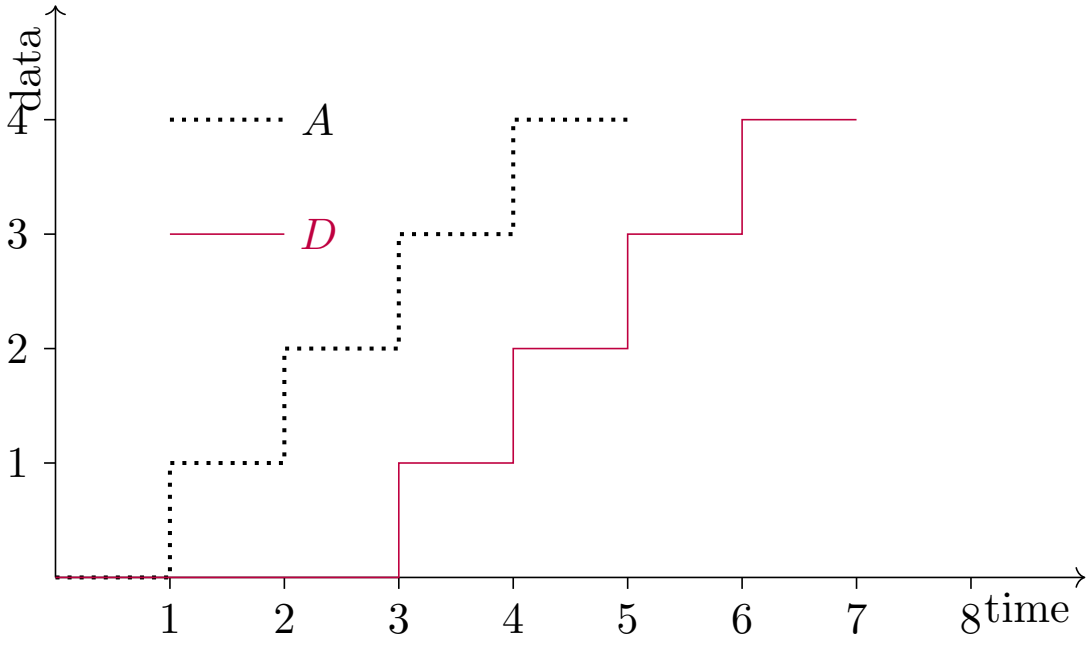

Figure 4: Delay or loss

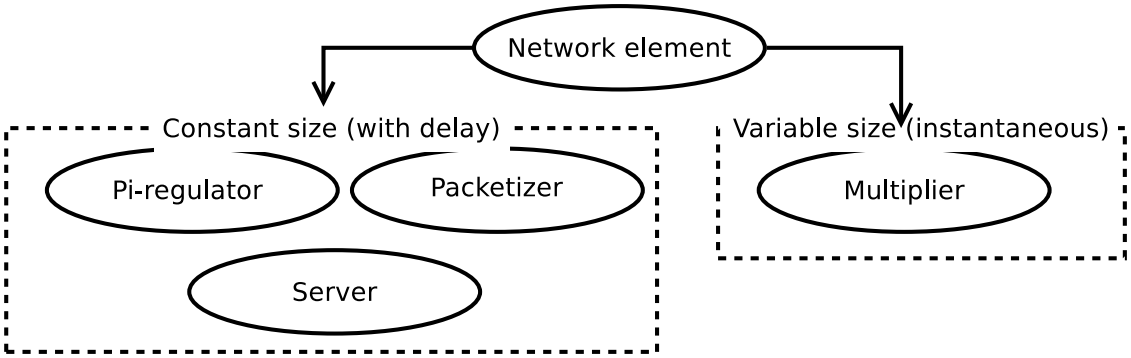

Figure 5: Illustration of the hierarchy of the network elements in Network Calculus.

received at time 8, with an overhead of 12 data units, and forwarding at time 11 the packet received at time 9, with an overhead of 6 data units.

However, the common constraint that $C \geq D$ is violated, and no simple definition of delay appears. The first aim of this paper is to give a definition that can capture such behaviour.

Another situation is depicted in Figure 4. It can represent either a system that introduces a delay of 2 time units before forwarding data, or a system that has dropped the two first units of data (multipliers can model dropping elements by setting null size to a bunch of data) and instantaneously forward the rest.

It illustrates the fact that it is hard to handle both data multiplying and delays in the same element. Then, it is easier to consider that a multiplier introduces no delay. If a real system encapsulating data introduces some delay, it will be modelled as the concatenation of a delay and a multiplier.

In the following, we will then define the two types of network elements and introduce the elements as the multiplier (inspired by [3]), the servers and the packetizers. Also, some new results on the packetizer, a server which waits for the entire bit of a packet before delivering it, are presented.

6.2 Multiplier definition

The notion of multiplier is inspired from the scaling servers from [3]. However, to go further with this notion as explain in this following, we need to adapt the notation and results.

Definition 8 (Multiplier)**.** *A multiplier M is a size-variant element such that $\forall (A, D) \in M$,*

$$\forall t, t' \in \mathbb{R}^+ : A(t) = A(t') \implies D(t) = D(t'). \tag{2}$$

The condition in eq. (2) implies that there is no data output if there is no data input during the same time interval.

Then, we can introduce the multiplier function as followed.

Definition 9 (Multiplier function)**.** *Let M be a multiplier, and $(A, D) \in M$. Then, a* multiplier function *for (A, D) is a non-decreasing function $M_{A,D} : \mathbb{R}^+ \to \mathbb{R}^+$ such that*

$$\forall t : M_{A,D}(A(t)) = D(t).$$

This definition is slightly different from the one of [3]: "a scaling function S assigns an amount of scaled data $S(a)$ to an amount of data a."

The first reason is related to non-determinism: a scaling element may have different outputs, depending on the input. For example, the CAN MAC layer introduces one bit of opposite value after a sequence of five equal bits. If C denotes such a layer, then, for a given input A (representing some amount of bits arrivals at each instant), there may exist several departure D, D', depending on the values of the bit sequence, not only on the amount.

The second reason is related to math: giving A and D is not sufficient to define a function: if A is not continuous, it will exists some $a \in \mathbb{R}^+$ such that there is not t with $A(t) = a$, and then, $S(a)$ is not defined.

Finally, we chose to rename the "scaling" as "multiplier" to avoid any confusion with the server (we use to abbreviate the network element with the first letter) and because we slightly change the definition of the network element.

Then, we can define the multiplier curves.

Definition 10 (Multiplier curves)**.** *Let M be a multiplier. Then, a function $\overline{\mu} \in \mathcal{F}^{\uparrow}$ (resp. $\underline{\mu} \in \mathcal{F}^{\uparrow}$) is a maximal (resp. minimal) multiplier curve if $\forall t, d \in \mathbb{R}^+, \forall (A, D) \in M : \underline{\mu}(A(t+d) - A(t)) \leq D(t+d) - D(t) \leq \overline{\mu}(A(t+d) - A(t))$.*

Then, we assume here that the multiplier is instantaneous.

Postulate 1 (No delay for multipliers)**.** *The delay of multipliers is zero.*

Remark 4. *The reason is given in Section 6.1. Thus, if a network element change the size and introduces a delay, is will be modelled as a sequence composed of a server and a multiplier.*

Some properties are also useful concerning the multiplier as the canonical multiplier, Property 5.

Property 5 (Canonical multiplier)**.** *Let f be a super-additive function. Then, the network element that associates $f(A)$ to any arrival A is a multiplier with minimal multiplier curve f.*

The proof only consists in applying the definition of a super-additive function on f.

Property 6 (Sequence of multipliers)**.** *A sequence of multipliers M, M' is also a multiplier, denoted $M; M'$. If M (resp. M') has minimal and maximal multiplier curves $\overline{\mu}, \underline{\mu}$ (resp. $\overline{\mu}', \underline{\mu}'$), then $M; M'$ has minimal and maximal multiplier curves $\overline{\mu}' \circ \overline{\mu}$ and $\underline{\mu}' \circ \underline{\mu}$.*

The proof only consists in applying the definition of a multiplier.

One of the good properties of the Network Calculus theory is that it supports well the aggregation of flows. Then, to be able to do that with a multiplier, we introduce a property on the aggregate flows passing through them.

Theorem 4 (Multiplier of aggregate flow)**.** *Let M_1, M_2 be two multiplier of respective minimal and maximal multiplier curves $\underline{\mu}_1, \underline{\mu}_2, \overline{\mu}_1, \overline{\mu}_2$. Then, the aggregate multiplier M_{12} that associates to each input $A_1 + A_2$ an output $D_1 + D_2$ such that $A_i \xrightarrow{M_i} D_i$ admits $\overline{\mu}_1 * \overline{\mu}_2$ and $\underline{\mu}_1 \overline{*} \underline{\mu}_2$ as minimal and maximal multiplier curves.*

Proof. By definition of multiplier, we have $\forall t, d \in \mathbb{R}^+$

$$\begin{aligned} &\underline{\mu}_1(A_1(t+d) - A_1(t)) \leq D_1(t+d) - D_1(t) \leq \overline{\mu}_1(A_1(t+d) - A_1(t)) \\ &\underline{\mu}_2(A_2(t+d) - A_2(t)) \leq D_2(t+d) - D_2(t) \leq \overline{\mu}_2(A_2(t+d) - A_2(t)) \end{aligned}$$

$$\begin{aligned} \implies \underline{\mu}_1(A_1(t+d) - A_1(t)) + \underline{\mu}_2(A_2(t+d) - A_2(t)) &\leq (D_1 + D_2)(t+d) - (D_1 + D_2)(t) \\ &\leq \overline{\mu}_1(A_1(t+d) - A_1(t)) + \overline{\mu}_2(A_2(t+d) - A_2(t)). \end{aligned} \tag{3}$$

Now, by definition of the min-plus and max-plus convolution, $\forall f, g \in \mathcal{F}$ and $u, v \in \mathbb{R}^+$

$$(f * g)(u+v) \leq f(u) + g(v) \leq (f \overline{*} g)(u+v).$$

The, eq. (3) leads to

$$\begin{aligned} \implies &(\underline{\mu}_1 * \underline{\mu}_2)(A_1(t+d) - A_1(t) + A_2(t+d) - A_2(t)) \leq (D_1 + D_2)(t+d) - (D_1 + D_2)(t) \leq \\ &\qquad (\overline{\mu}_1 \overline{*} \overline{\mu}_1)(A_1(t+d) - A_1(t) + A_2(t+d) - A_2(t)) \\ \iff &(\underline{\mu}_1 * \underline{\mu}_2)((A_1 + A_2)(t+d) - (A_1 + A_2)(t)) \leq (D_1 + D_2)(t+d) - (D_1 + D_2)(t) \leq \\ &\qquad (\overline{\mu}_1 \overline{*} \overline{\mu}_1)((A_1 + A_2)(t+d) - (A_1 + A_2)(t)). \end{aligned}$$

□

Finally, adapted from [3, Theorem 3.1], we rewrite the inversion of the server and the multiplier. The aim is to be able to regroup on one side all the multipliers together and on the other side all the servers to be able to compute the delay using for instance the Pay Burst Only Once (PBOO, [17]).

Theorem 5 (Inverting multiplier and server)**.** *Let S be a server offering a minimal min-plus service of function β_m and a maximal service curve β^M and shaping curve σ. Also, let M be a multiplier, with a minimal multiplier curve $\underline{\mu}$ and a maximal multiplier curve $\overline{\mu}$. And let A, B, and C be respective input/output of the sequence: $A \xrightarrow{S} B \xrightarrow{M} C$.*

If $\underline{\mu}$ is super-additive and $\overline{\mu}$ is sub-additive, then it exists B' and S' such that $A \xrightarrow{M} B' \xrightarrow{S'} C$ where S' offers a minimal min-plus service of function $\underline{\mu}(\beta_m)$ and a maximal service curve $\overline{\mu}(\beta^M)$ and a shaping curve $\overline{\mu}(\sigma)$.

Figure 6 illustrates the result of Theorem 5: the inversion of the server and the multiplier. Under the assumptions, the left sequence can be replaced by the right sequence.

Such result was previously presented in [3], but the proof was using some false argument (discussed in Appendix 9.1), and the formal definition of scaling/multipliyer was slightly different (for reasons given in Section 6.1),leading to a new proof. In [3], the inversion is possible also in the opposite way, assuming that the scaling function is bijective, an assumption that is not done in this work (cf. Section 6.1).

Figure 6: Theorem 5 allows replacing the left part with the right part.

Proof. Looking for the shaping curve, according to Definition 3, we have that $\forall t, d \in \mathbb{R}^+$, $C(t+d) - C(t) \leq \overline{\mu}(B(t+d) - B(t)) \leq \overline{\mu}(\sigma(d))$. Consequently, it is equivalent to the system where A passes thourgh a multiplier M and a shaper $\overline{\mu}(\sigma)$.

Now, we will focus on the minimal min-plus and maximal services. Using Definition 3, we have

$$A * \beta_m \leq B, \qquad A * \beta^M \geq B.$$

Also, using Definition 10 (with $t = 0$), we have

$$\underline{\mu}(B) \leq C, \qquad \overline{\mu}(B) \geq C.$$

As $\underline{\mu}$ and $\overline{\mu}$ are nondecreasing, we can combine the results:

$$\underline{\mu}(A * \beta_m) \leq \underline{\mu}(B) \leq C, \quad \overline{\mu}(A * \beta^M) \geq \overline{\mu}(B) \geq C.$$

Then, if $\underline{\mu}$ is super-additive and $\overline{\mu}$ is sub-additive, we can use Property 1 and we have

$$\begin{cases} (\underline{\mu} \circ A) * (\underline{\mu} \circ \beta_m) \leq \underline{\mu}(A * \beta_m) \leq \underline{\mu}(B) \leq C, \\ (\overline{\mu} \circ A) * (\overline{\mu} \circ \beta^M) \geq \overline{\mu}(A * \beta^M) \geq \overline{\mu}(B) \geq C. \end{cases}$$

Consequently, it corresponds to pass through a multiplier with a minimal multiplier curve $\underline{\mu}$ and a maximal multiplier curve $\overline{\mu}$ and a server S' which offers a minimal min-plus service of function $\underline{\mu}(\beta_m)$ and a maximal service curve $\overline{\mu}(\beta^M)$ and a shaping curve $\overline{\mu}(\sigma)$. □

7 Use of results

The aim of this section is to compare the results of the previous modelization in Network Calculus with the results of the CPA theory. To do that, we will analyse a system illustrated in Figure 7 composed of:

- CPU1: a Static Priority Preemptive CPU with 3 periodic tasks $(T1, T2, T3)$ in ascending order of priority.
- CPU2: a Static Priority Preemptive CPU with 3 periodic tasks $(T4, T5)$ in ascending order of priority.
- CAN Bus: a Static Priority Non Preemptive bus with 3 release tasks $(T'2, T'3, T'4)$ respectively released by $(T2, T3, T4)$ in ascending order of priority.
- CPU3: a Static Priority Preemptive CPU with 3 released tasks $(T''2, T''3, T''4)$ respectively released by $(T'2, T'3, T'4)$ in ascending order of priority.

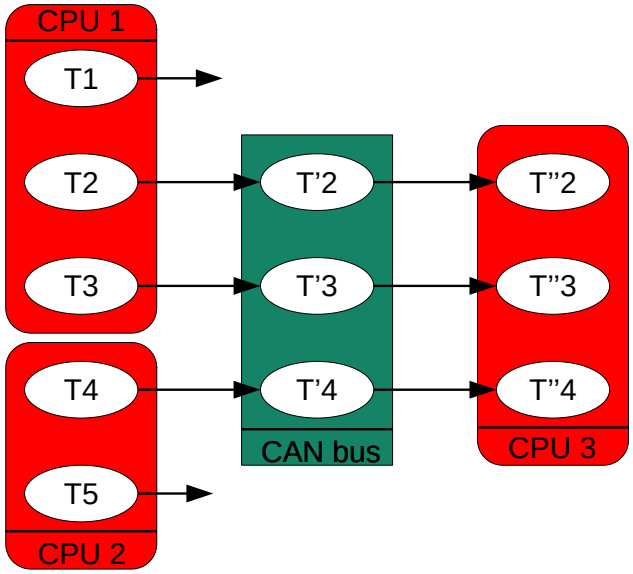

Figure 7: Illustration of the system of the case study.

Table 1: Parameters for the case study (ms)

Task	**Period**	**Processing time**	**Task**	**Processing time**
$T1$	15	3	$T2'$	4
$T2$	30	4	$T3'$	8
$T3$	45	15	$T4'$	2
$T4$	60	5	$T2''$	2
$T5$	120	10	$T3''$	4
			$T4''$	1

The strength of the Network Calculus is to be able to aggregate flows. Here, this example is interesting thanks to the possibility of aggregating $T2$ and $T3$ to compute an upper bound on the delay of $T4$.

We will use a set of parameters detailed in Table 1.

The aim is to calculate the worst-case execution times of the three task chains: C2: $T2 \rightarrow T2' \rightarrow T2''$, C3: $T3 \rightarrow T3' \rightarrow T3''$, and C4: $T4 \rightarrow T4' \rightarrow T4''$.

This system will be analyzed with 1) the CPA method (the results will be noted *CPA*), with 2) current Network Calculus methods, *i.e.* considering SFA and TFA combined with [3] and keeping the minimum) (the results will be noted *NC*) and 3) with the same Network Calculus algorithms plus our results Note that as the tasks $T2$ and $T3$ come from the same source and have the same route, the Network Calculus allows us to compute the worst-case execution time considering the aggregate flow composed of these two tasks.

The results of the three analyses are given in Table 2. The code is given in Appendix B. In

this case, the TFA algorithm gives always the smaller bounds, but no generic conclusion can be derived from this simple case (cf. [23]),

Table 2: Results of the analyse of the case study.

Task chain	**Worst-case bounds (ms)**		
	CPA	NC	NC-agg
C2	21	21	21
C3	45	45	45
C4	26	30	23.2

First, we note that the worst case execution time of the task chains $C1$ and $C2$ are the same regardless of the analysis used. However, we see a difference with the task chain $C4$. As we said, one of the pluses of the Network Calculus theory is the easy handling of aggregate flows. The shaping on the tasks $T2$ and $T3$ and the released one allows the Network Calculus to improve the results of the analysis. Note that without the aggregation, the Network Calculus cannot improve the results and is more pessimistic than CPA.

8 Conclusion

In this paper, we present novel results concerning packetization/scaling and flow aggregation within the Network Calculus framework. These results were motivated by the objective of enhancing the bounds of released task chains, considering the inherent limitation that a subsystem cannot transmit more data than it can process. Additionally, we demonstrate the efficacy of our results through an illustrative example, showcasing the benefits of flow aggregation in Network Calculus.

References

[1] T.-S. N. T. Group, "Ieee standards 802.1," Tech. Rep., Jul. 6 2016. [Online]. Available: https://www.ieee802.org/1/pages/tsn.html

[2] P. Roux, S. Quinton, and M. Boyer, "A formal link between response time analysis and network calculus," in *34th Euromicro Conference on Real-Time Systems ECRTS 2022*, 2022.

[3] M. Fidler and J. B. Schmitt, "On the way to a distributed systems calculus: An end-to-end network calculus with data scaling," in *Proc. of the Int. Conf. on Measurement and modeling of computer systems*, 2006, pp. 287–298.

[4] A. Bouillard, L. Jouhet, and E. Thierry, "Service curves in Network Calculus: dos and don'ts," INRIA, Research Report RR-7094, 2009. [Online]. Available: http://hal.inria.fr/inria-00431674/PDF/RR-7094.pdf

[5] M. Boyer, E. Le Corronc, and A. Bouillard, *Deterministic network calculus: From theory to practical implementation.* John Wiley & Sons, 2018.

[6] E. Wandeler, A. Maxiaguine, and L. Thiele, "Quantitative characterization of event streams in analysis of hard real-time applications," *Real-Time Systems*, vol. 29, no. 2-3, pp. 205–225, 2005.

[7] E. Wandeler and L. Thiele, "Characterizing workload correlations in multi processor hard real-time systems." in *Proc. of the 11th IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS 2005)*, 2005, pp. 46–55.

[8] J. Schlatow and R. Ernst, "Response-time analysis for task chains in communicating threads," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2016, pp. 1–10.

[9] ——, "Response-time analysis for task chains in communicating threads," in *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016, pp. 1–10.

[10] M. G. Harbour, M. H. Klein, and J. P. Lehoczky, "Fixed priority scheduling periodic tasks with varying execution priority," in *RTSS*. Citeseer, 1991, pp. 116–128.

[11] Z. A. H. Hammadeh, R. Ernst, S. Quinton, R. Henia, and L. Rioux, "Bounding deadline misses in weakly-hard real-time systems with task dependencies," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, 2017, pp. 584–589.

[12] A. Girault, C. Prévot, S. Quinton, R. Henia, and N. Sordon, "Improving and estimating the precision of bounds on the worst-case latency of task chains," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 37, no. 11, pp. 2578–2589, 2018.

[13] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocess. Microprogram*, vol. 40, no. 2–3, pp. 117–134, 1994.

[14] S. Künzli, A. Hamann, R. Ernst, and L. Thiele, "Combined approach to system level performance analysis of embedded systems," in *Proc. of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*. ACM, 2007, pp. 63–68.

[15] L. Köhler, B. Nikolic, R. Ernst, and M. Boyer, "Increasing accuracy of timing models: from cpa to cpa+," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1210–1215.

[16] M. Boyer and D. Doose, "Combining network calculus and scheduling theory to improve delay bounds," in *Proceedings of the 20th International Conference on Real-Time and Network Systems*, 2012, pp. 51–60.

[17] J.-Y. Le Boudec and P. Thiran, Eds., *Network Calculus*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 3–81. [Online]. Available: https://doi.org/10.1007/3-540-45318-0_1

[18] M. Boyer and P. Roux, "Embedding network calculus and event stream theory in a common model," in *2016 IEEE 21st Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, 2016, pp. 1–8.

[19] D. Guidolin--Pina and M. Boyer, "Looking for equivalences of the services between left and right continuity in the Network Calculus theory," Sep. 2022, working paper or preprint. [Online]. Available: https://hal.science/hal-03772867

[20] R. L. Cruz and C. Okino, "Service guarantees for window flow control," in *Proc. of the Annual Allerton Conf. on Communication Control and Computing*, vol. 34. Citeseer, 1996, pp. 10–21.

[21] R. Cruz, "Quality of service guarantees in virtual circuit switched networks," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 6, pp. 1048–1056, 1995.

[22] J.-Y. Le Boudec, "A theory of traffic regulators for deterministic networks with application to interleaved regulators," *IEEE-ACM Transactions On Networking*, vol. 26, no. 6, pp. 2721–2733, 2018.

[23] M. Boyer, A. Graillat, B. Dupont De Dinechin, and J. Migge, "Bounding the delays of the MPPA network-on-chip with network calculus: models and benchmarks," *Performance Evaluation*, July 2020.

9 Appendix

9.1 Continuity issue in the proof of the inversion of a server and a multiplier

As mentioned after Theorem 5, the third equation of the proof of [3, Thm. 3.1] can be wrong due to a continuity issue. Let us remind the equation: S, F, and β are three nondecreasing functions and t a positive real, then, the proofs states that

$$S\left(\inf_{0\leq s\leq t}\{F(s)+\beta(t-s)\}\right)=\inf_{0\leq s\leq t}\{S\left(F(s)+\beta(t-s)\right)\}.$$

However, this equation can be wrong depending in continuity. Here is a counter-example. Consider $\beta(t)=t$,

$$S(t)=\begin{cases}0 & \text{if } t\leq 1\\ 1 & \text{otherwise}\end{cases}, F(t)=\begin{cases}0 & \text{if } t<2\\ 3 & \text{otherwise}\end{cases}.$$

We will compute both sides of the equation with $t=3$.

1. left side: it is equal to $S(F*\beta)(3)=0$.
2. right side: First, note that, as S is nondecreasing, the infimum for $s\in[0,3]$ of the expression $S\left(F(s)+\beta(3-s)\right)$ will be reached at the same point than the expression inside the function S: $F(s)+\beta(3-s)$.

   We then need to compute the infimum of this latter expression:

$$F(s)+\beta(3-s)=\begin{cases}3-s & \text{if } 0\leq s<2\\ 6-s & \text{if } 2\leq s\leq 3\end{cases}$$

As a consequence, $F(s)+\beta(3-s) > 1$, but the infimum tends towards 1, by above. However, if $t > 1, S(t) = 1$. Then, $\inf_{0 \leq s \leq t} \{S(F(s)+\beta(t-s))\} = 1$ which is different from the result of the left side.

9.2 Network Calculus code to compute delay bounds on the case study with and without the results of this paper

The code presented in Listing 1 give the delay bounds computed with the SFA and TFA algorithms for the case study introduced Section 7. It can be run online[3].

```
// Periods of tasks
P_T1 := 15
P_T2 := 30
P_T3 := 45
P_T4 := 60
P_T5 := 120

// Workloads (p means ') of tasks
C_T1 := 3
C_T2 := 4
C_T3 := 15
C_T4 := 5
C_T5 := 10

C_T2p := 4
C_T3p := 8
C_T4p := 2

C_T2pp := 2
C_T3pp := 4
C_T4pp := 1

// Maximum arrival curves for input tasks
aT1 := stair(0, P_T1, C_T1)
aT2 := stair(0, P_T2, C_T2)
aT3 := stair(0, P_T3, C_T3)
aT4 := stair(0, P_T4, C_T4)
aT5 := stair(0, P_T5, C_T5)
aT23 := aT2 + aT3

// Service curves for all the servers
beta_m := affine(1,0)
beta_M := affine(1,0)
sigma := affine(1,0)

// Residual service curve in the CPU1/2 (Static Priority Preeptive)
```

[3] https://www.realtimeatwork.com/minplus-playground

```
b_r_T1 := beta_m
b_r_T2 := nnupclosure(beta_m - aT1)
b_r_T3 := nnupclosure(beta_m - aT1 - aT2)
b_r_T4 := beta_m
b_r_T5 := nnupclosure(beta_m - aT4)
b_r_T23 := nnupclosure(beta_m - aT1)

// Local delays of the first server (CPU1 or CPU2 depends on the task)
delay_T1 := hDev(aT1, b_r_T1)
delay_T1
delay_T2 := hDev(aT2, b_r_T2)
delay_T2
delay_T3 := hDev(aT3, b_r_T3)
delay_T3
delay_T4 := hDev(aT4, b_r_T4)
delay_T4
delay_T5 := hDev(aT5, b_r_T5)
delay_T5
delay_T23 := hDev(aT23, b_r_T23)
delay_T23

// Output maximal arrival curve before the scaling (Server + Packetizer)
aT2p_aS := ((aT2 * beta_M) / (b_r_T2 - C_T2))
     /\ (sigma + C_T2) /\ (aT2 / delay(delay_T2))
aT3p_aS := ((aT3 * beta_M) / (b_r_T2 - C_T3))
     /\ (sigma + C_T3) /\ (aT3 / delay(delay_T3))
aT4p_aS := ((aT4 * beta_M) / (b_r_T2 - C_T4))
     /\ (sigma + C_T4) /\ (aT4 / delay(delay_T4))
aT23p_aS := ((aT23 * beta_M) / (b_r_T23 - C_T2 - C_T3))
     /\ (sigma + C_T2 + C_T3) /\ (aT23 / delay(delay_T2 /\ delay_T3))

// Scaling functions between CPU1/2 and BUS
s2 := affine(C_T2p/C_T2, 0)
s3 := affine(C_T3p/C_T3, 0)
s4 := affine(C_T4p/C_T4, 0)
s23 := (s2 * s3)

// Output maximum arrival curves after the scaling
aT2p := s2 comp aT2p_aS
aT3p := s3 comp aT3p_aS
aT4p := s4 comp aT4p_aS
aT23p := s23 comp aT23p_aS

// Residual service curve in the BUS (Static Priority NON Preeptive)
b_r_T2p := nnupclosure(beta_m - (C_T3p \/ C_T4p))
b_r_T3p := nnupclosure(beta_m - aT2p - C_T4p)
b_r_T4p := nnupclosure(beta_m  - aT2p - aT3p)
b_r_T4p_aggr := nnupclosure(beta_m  - aT23p)
```

```
b_r_T23p := nnupclosure(beta_m - C_T4p)

// Local delay of the second server (BUS)
delay_T2p := hDev(aT2p, b_r_T2p)
delay_T2p
delay_T3p := hDev(aT3p, b_r_T3p)
delay_T3p
delay_T4p := hDev(aT4p, b_r_T4p)
delay_T4p
delay_T4p_aggr := hDev(aT4p, b_r_T4p_aggr)
delay_T4p_aggr
delay_T23p := hDev(aT23p, b_r_T23p)
delay_T23p

// Output maximal arrival curve before the scaling (Server + Packetizer)
aT2pp_aS := ((aT2p * beta_M) / (b_r_T2p - C_T2p))
     /\ (sigma + C_T2p) /\ (aT2p / delay(delay_T2p))
aT3pp_aS := ((aT3p * beta_M) / (b_r_T2p - C_T3p))
     /\ (sigma + C_T3p) /\ (aT3p / delay(delay_T3p))
aT4pp_aS := ((aT4p * beta_M) / (b_r_T2p - C_T4p))
     /\ (sigma + C_T4p) /\ (aT4p / delay(delay_T4p))
aT23pp_aS := ((aT23p * beta_M) / (b_r_T23p - C_T2p - C_T3p))
     /\ (sigma + C_T2p + C_T3p) /\ (aT23p / delay(delay_T2p /\ delay_T3p))

// Scaling functions between BUS and CPU3
s2p := affine(C_T2pp/C_T2p, 0)
s3p := affine(C_T3pp/C_T3p, 0)
s4p := affine(C_T4pp/C_T4p, 0)

// Output maximum arrival curves after the scaling
aT2pp := s2p comp aT2pp_aS
aT3pp := s3p comp aT3pp_aS
aT4pp := s4p comp aT4pp_aS
aT23pp := (s2p * s3p) comp aT23pp_aS

// Residual service curve in the CPU3 (Static Priority Preeptive)
b_r_T2pp := nnupclosure(beta_m)
b_r_T3pp := nnupclosure(beta_m - aT2pp)
b_r_T4pp := nnupclosure(beta_m  - aT2pp - aT3pp)
b_r_T4pp_aggr := nnupclosure(beta_m  - aT23pp)
b_r_T23pp := nnupclosure(beta_m)

// Local delay of the third server (CPU3)
delay_T2pp := hDev(aT2pp, b_r_T2pp)
delay_T2pp
delay_T3pp := hDev(aT3pp, b_r_T3pp)
delay_T3pp
delay_T4pp := hDev(aT4pp, b_r_T4pp)
```

```
delay_T4pp
delay_T4pp_aggr := hDev(aT4pp, b_r_T4pp_aggr)
delay_T4pp_aggr
delay_T23pp := hDev(aT2pp, b_r_T23pp)
delay_T23pp

// Global delays (sum of local delay)
delay_total_T2 := delay_T2 + delay_T2p + delay_T2pp
delay_total_T2
delay_total_T3 := delay_T3 + delay_T3p + delay_T3pp
delay_total_T3
delay_total_T4 := delay_T4 + delay_T4p + delay_T4pp
delay_total_T4
delay_total_T23 := delay_T23 + delay_T23p + delay_T23pp
delay_total_T23

// Global delay (aggregation of 2/3 to compute 4)
delay_total_T4_aggr := delay_T4 + delay_T4p_aggr + delay_T4pp_aggr
delay_total_T4_aggr

// Global delays (PBOO)
b_T2 := (b_r_T2 - C_T2) \/ 0
b_T2p := (b_r_T2p - C_T2p) \/ 0
b_T2pp := (b_r_T2pp - C_T2pp) \/ 0
beta_PBOO_T2 := ((s2 * s2p) comp b_T2) * (s2p comp b_T2p) * b_T2pp
delay_PBOO_T2 := hDev(((s2 * s2p) comp aT2), beta_PBOO_T2)
delay_PBOO_T2
b_T3 := (b_r_T3 - C_T3) \/ 0
b_T3p := (b_r_T3p - C_T3p) \/ 0
b_T3pp := (b_r_T3pp - C_T3pp) \/ 0
beta_PBOO_T3 := ((s3 * s3p) comp b_T3) * (s3p comp b_T3p) * b_T3pp
delay_PBOO_T3 := hDev(((s3 * s3p) comp aT3), beta_PBOO_T3)
delay_PBOO_T3
b_T4 := (b_r_T4 - C_T4) \/ 0
b_T4p := (b_r_T4p - C_T4p) \/ 0
b_T4pp := (b_r_T4pp - C_T4pp) \/ 0
beta_PBOO_T4 := ((s4 * s4p) comp b_T4) * (s4p comp b_T4p) * b_T4pp
delay_PBOO_T4 := hDev(((s4 * s4p) comp aT4), beta_PBOO_T4)
delay_PBOO_T4
b_T4_aggr := (b_r_T4 - C_T4) \/ 0
b_T4p_aggr := (b_r_T4p_aggr - C_T4p) \/ 0
b_T4pp_aggr := (b_r_T4pp_aggr - C_T4pp) \/ 0
beta_PBOO_T4_aggr := ((s4 * s4p) comp b_T4_aggr)
                     * (s4p comp b_T4p_aggr) * b_T4pp_aggr
delay_PBOO_T4_aggr := hDev(((s4 * s4p) comp aT4), beta_PBOO_T4_aggr)
delay_PBOO_T4_aggr
```

Listing 1: Network Calculus code to compute delay bounds on the case study