This regular paper was presented as part of the main technical program at IEEE ETFA'2011

# Non preemptive static priority with network calculus

William Mangoua Sofack, Marc Boyer
ONERA
2, av E. Belin, Toulouse, France
William.Mangoua_Sofack@onera.fr, Marc.Boyer@onera.fr

## Abstract

*The paper addresses performance analysis of embedded real time networks. We use network calculus to assess the quality of service of a residual flow in a context of aggregation with non preemptive fixed priority scheduling. The main contribution concerns the evaluation of the residual service, given to the low priority flows. In previous works, the effect of the non-preemption seems to be not well taken into account. In particular, when a non preemptive flow is served, it benefits from the full speed from the server, even if, from a long term point of view, it gets only a fractional part. The modeling of this aspect is the starting point of this work*[1].

## 1. Introduction

Embedded networks are increasingly communicating networks. For example, in a modern aircraft, there are more than hundred computers connected to the avionic backbone with several applications that exchange informations in real time. Obviously, a correct behaviour of these applications must be guaranteed. From the point of view of the network, this means that any message transmitted over the network must arrive at its destination on time. This implies then to have a good measure of the end-to-end delay (known as Worst Case Traversal Time, WCTT).

The network calculus theory [5, 1, 3] provides a formal framework for modeling of communication networks and computing such WCTT. The aim of this work is to reduce the pessimism of the method in a specific case.

We consider a node shared by several flows, with a *non preemptive static priority policy*. We are interested in the residual service left by the higher priority ones to lower priority ones. This policy has been already studied in [1, 2, 8], but these results appear not to carefully consider the hypothesis of non-preemption. In particular, when a non preemptive flow is served, it benefits from the full speed from the server, even if, from long term point of view, it gets only a fractional part. This assessment was also the starting point of [4], but it leaves two open questions: is there any analytical formulation of their result? And is the result tight? In this paper, an analytical expression (with the proof) is given, in a more general context, and an example showing the non-tightness of our result is given.

Initially, network calculus is presented in Section 2.1, and the previous works [1, 2, 8, 4] are detailed in Section 2.2. This presentation is necessary to understand our contribution, summarised in Section 2.3. Our modelling of non preemptive static priority policy in network is presented in Section 3. The technical presentation of our contribution can then be done (Section 4), with the Theorem 4.1 considering 3 priority levels, and Corollary 4.6 generalising the result to any number of priority. Our approach is illustrated by examples and results are compared to existing ones (Section 5). Section 6 concludes.

## 2. State of the art

Analysis of the time-related performance during design of embedded systems has been approached from various angles, such as scheduling analysis, completion time analysis and model checking in timed automata with limitations inherent in their fields [9]. Network Calculus is a theory of deterministic queuing systems, developed to compute worst end-to-end communication bounds in networks [5, 6, 1].

### 2.1. Network Calculus

The network calculus analysis focuses on worst case performances. The information about the system features are stored in functions, such as arrival curves characterising the traffic or service curves quantifying the service guaranteed at the network nodes. These functions can be combined together thanks to special network calculus operations, to get bounds on buffers size or delays.

#### 2.1.1 Mathematical background: $(\min, +)$ dioid

Here are presented some operators of the $(\min, +)$ dioid used by network calculus. Beyond usual operations like the minimum or the addition of functions, network calculus makes use of several classical operations which

[1]This works has been partialy funded by French ANR agency under project id ANR-09-SEGI-009.

978-1-4577-0018-7/11/$26.00 

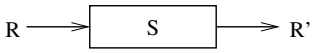


**Figure 1. Single flow input/output system.**

are the translations of $(+, \times)$ filtering operations into the $(\min, +)$ setting, as well as a few other transformations.

Network calculus mainly uses non-decreasing functions, and related operators. Here are those used in this article.

**Set** $\mathcal{F}$ $\mathcal{F}$ denote the set of wide-sense increasing functions $f : \mathbb{R} \to \mathbb{R} \cup \{+\infty\}$ such that $f(t) = 0$ for $t < 0$.

**Function** $[\,]^+ : x \mapsto \max(x, 0)$.

**Vertical deviation** It is defined for two functions $f$ and $g$ by $v(f, g) = \sup_{t \geq 0} \{f(t) - g(t)\}$

**Horizontal deviation** It is defined for two functions $f$ and $g$ by $h(f, g) = \sup_{t \geq 0} \{\inf \{d \geq 0 \mid f(t) \leq g(t + d)\}\}$

**Min-plus convolution** It is defined for two functions $f$ and $g$ by $(f * g)(t) = \inf_{0 \leq s \leq t} \{f(t - s) + g(s)\}$

**Positive and non-decreasing upper closure** It is defined for a functions $f$ by $f \uparrow (t) = [\sup_{0 \leq s \leq t} f(s)]^+$

**The pseudo inverse** $f^{-1}$ of the function $f$ is defined by

$$f^{-1}(u) = \inf\{t | f(t) \geq u\} \tag{1}$$

This implies, if $u \leq f(t)$, then $t \geq f^{-1}(u)$. Moreover, if $f \in \mathcal{F}$, $u \geq f(t)$, then $t \leq f^{-1}(u)$.

**The strict pseudo inverse** $f^{str-1}$ of the function $f$ is defined by

$$f^{str-1}(u) = \inf\{t | f(t) > u\} \tag{2}$$

Then, $\forall t < f^{str-1}(u),\ f(t) \leq u$.
Plus, if $f(t) > u$ then, $t \geq f^{str-1}(u)$

### 2.1.2 Network calculus: reality modelling

A network calculus model for a communication network consists in the three following components:
A partition of the network into subsystems (often called nodes) which may have different scales (from elementary hardware like a processor to large sub-networks).
A description of data flows, where each flow follows a path through a specified sequence of subsystems and where each flow is shaped by some arrival curve just before entering the network.
A description of the behaviour of each subsystem, that is service curves bounding the performances of each subsystem, as well as service policies in case of multiplexing (several flows entering the same subsystem and thus sharing its service).

In network calculus, the real flows are modelled by cumulative functions $R \in \mathcal{F}$: $R(t)$ counts the total amount of data produced by the flow up to time $t$.

Consider a system $S$, which we view as a black box; $S$ receives a input flow, $R(t)$, and delivers the data after a variable delay. Call $R'(t)$ the output flow: $R \xrightarrow{S} R'$. We have relation $R' \leq R$, meaning that data goes out after being entered. System $S$ might be, for example, a single buffer served at a constant rate, a complex communication node, or even a complete network. Figure 1 shows input and output functions for a single server queue.

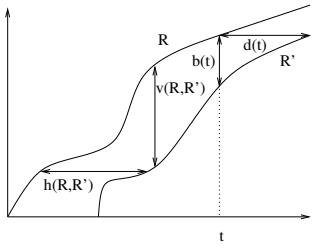


**Figure 2. Backlog and delay**

The main network calculus performance measures are backlog and delay (Figure 2).

The *backlog* is the amount of bits that are held inside the system; if the system is a single buffer, it is the queue length. In contrast, if the system is more complex, then the backlog is the number of bits "in transit", assuming that we can observe input and output simultaneously [1]. For a system where $R$ is the input and $R'$ the output, *the backlog at time* $t$ *is* $b(t) = R(t) - R'(t)$. Obviously, $b(t) \leq v(R, R')$.

A *backlogged period* is a period during which the backlog is not zero. Let $t$, a moment in a backlogged period, this backlogged period has started at $StBl(t) = \sup \{u \leq t \mid R'(u) = R(u)\}$. We limit the following to left-continuous functions to ensure:

$$R'(StBl(t)) = R(StBl(t)) \tag{3}$$

The *virtual delay* at a time $t$ is the delay that a bit entered at time $t$ will wait until going out, defined by $d(t) = \inf \{\tau \geq 0 \mid R(t) \leq R'(t + \tau)\}$. Obviously $d(t) \leq h(R, R')$.

### 2.1.3 Network calculus: contract modeling

To provide guarantees to data flows, one need to know some traffic contract on the traffics and the services in the network. For this purpose, network calculus provides the concepts of arrival curve and service curve.

**Arrival curve** A flow $R \in \mathcal{F}$ *is constrained by* $\alpha \in \mathcal{F}$ if and only if for all $s \leq t$: $R(t) - R(s) \leq \alpha(t - s)$. We say also that $R$ *has* $\alpha$ *as an arrival curve*, or also that $R$ *is* $\alpha$*-smooth*. This condition is equivalent to $R \leq R * \alpha$.

**Service curve** The behaviour of a server is modelled by the concept of service curve, modelling some guarantees on the service provided to flows.

The literature offers several definitions for different flavors of service. [2] proposes a comparative study. Consider a system $R \xrightarrow{S} R'$, *i.e.* a server $S$ with input $R$ and output $R'$ (Figure 1).

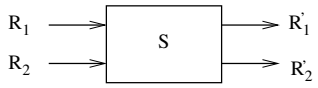


**Figure 3. Aggregation.**

We say that $S$ offers to the flow a *simple service of curve* $\beta$ if and only if $R' \geq R * \beta$. We say that a system $S$ offers a *strict service of curve* $\beta$ if, during any backlogged period $[t, s[$, we have $R'(s) - R'(t) \geq \beta(s-t)$. We say that a system $S$ offers a *strict service curve (weak sense)* $\beta$ to a flow if, $\forall t \geq 0$, $R'(t) - R'(StBl(t)) \geq \beta(t - StBl(t))$. Note that if $t$ is not a moment of a backlogged period, then $StBl(t) = t$.

Let us now present the main network calculus results:

**Theorem 2.1** (Backlog and delay bound)**.** *Assume a flow, constrained by an arrival curve $\alpha$, traverses a system that offers a service curve $\beta$, the backlog $b(t)$ for all $t$ satisfies: $b(t) \leq v(\alpha, \beta)$.*
*The virtual delay $d(t)$ for all $t$ satisfies: $d(t) \leq h(\alpha, \beta)$.*

#### 2.1.4 Aggregation, simple and strict services

Modelling aggregation is an important issue in network calculus. Aggregation means that the service is shared by different flows: for example, if a server $S$ offers an aggregated simple service of curve $\beta$ to two flows $R_1$ and $R_2$ ( $(R_1, R_2) \xrightarrow{S} (R'_1, R'_2)$, Figure 3), it means that it offers this service to the flow $R = R_1 + R_2$ (*i.e.* $R'_1 + R'_2 \geq (R_1 + R_2) * \beta$), but the distribution of the service between the flows depends on priority flows and server policy (common policies are FIFO, strict priorities...). Several results exist to derive the residual service $S_i$ offered to each flow ($R_i \xrightarrow{S_i} R'_i$). But the flavor of service is crucial: some results make assumption of strict service, others of simple. And the residual service can also be simple or strict.

These different notions of service are then of practical importance: the strict service is a stronger property, and some results only hold for strict service. In particular, when a hierarchy of scheduling policies is used (first, static priority is used, and the low level is decomposed into sub-flow, scheduled with another policy), having a strict residual service is sometime needed to handle the sub-flows.

### 2.2. Related works

This paper focuses on non preemptive static priorities, and the residual service of the low priority flows. In the following, in all presented results, $i < j$ implies that $R_i$ has higher priority than $R_j$.

Several results have been published on this subject, each one refining the previous results. They are presented in the sequel.

**Simple residual service** The initial results of [1, Cor. 6.2.1] claims that, if a server with strict service $\beta$ is shared by two flows $R_1$ and $R_2$, $R_1$ having arrival curve $\alpha_1$, then the low priority flow $R_2$ receives a minimal residual service of curve $[\beta - \alpha_1]^+$.

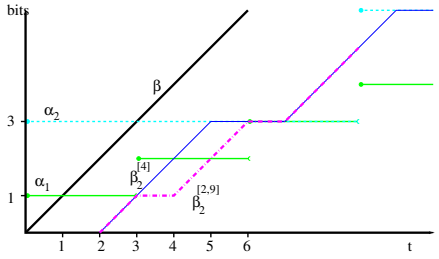


**Figure 4. Comparison between [2, 8] and [4].**

It has two limitations: the residual service is limited to non-decreasing results, and is not strict (see 2.1.4). The limitation to non non-decreasing results leads to the exclusion of some real cases [2].

**Strict residual service** The two problems described above have been independently studied by [2, 8] with exactly the same result[3].

Let then consider $n$ flows $R_1, \ldots, R_n$, each $R_i$ having an arrival curve $\alpha_i$ and a maximum packet size $l_i^{max}$, and still a strict service of curve $\beta$. Then, each flow $R_i$ receives the simple service of curve $\beta_i$ and the strict service of curve $\beta'_i$, defined by:

$$\beta_i = (\beta - \textstyle\sum_{j=1}^{i-1} \alpha_j - \max_{i<j<n} l_j^{max}) \uparrow \qquad (4)$$

$$\beta'_i = (\beta - \textstyle\sum_{j=1}^{i-1} \alpha_j - \max_{i\leq j<n} l_j^{max}) \uparrow \qquad (5)$$

Note that the only difference between both is that, in the strict service case, the flow $R_i$ competes versus its own maximal packet size (the term $-l_i^{\max}$), and it does not in the simple service one. In [2] it is shown on an example that this "self competitive term" can not be ignored in the strict case[4].

The results are not presented exactly in this way: [2, Cor. 2] presents the results without the non-decreasing upper closure, which can be added in the strict service case

[2] For example, if $\beta$ is a linear function, modelling a constant rate service, and $\alpha_1$ is a stair-case function, modelling a per-packet arrival, $\beta - \alpha_1$ is locally decreasing at each packet arrival

[3] At first glance, one should consider both results different, since one apply for network calculus and the other for real-time calculus. But as shown in [2], the strict minimal service of network calculus is equivalent to the service curve of RTC.

[4] This notion of "competition versus itself" is a bit counter-intuitive, and deserves an informal explanation. For example, with two priority levels, $\beta_2$ and $\beta'_2$ are given by $\beta_2 = (\beta - \alpha_1) \uparrow$ and $\beta'_2 = (\beta - \alpha_1 - l_2^{max}) \uparrow$. Consider a backlogged period of the low priority flow $R_2$. Before having the server, the flow $R_2$ must wait the end of service of the high-priority flow $R_1$ (the term $\alpha_1$). But due to the non-preemption, the high priority flow can have been blocked by one packet of the low priority flow. And this could be a *previous* backlogged period of the flow $R_2$, not the considered one, as in Figure 6 . This makes the difference between strict service (considering *any* backlogged period) and the simple service one, based on convolution, that is so say, *some* period.

| | Period | Size | $\alpha_i$ |
|---|---|---|---|
| $R_1$ | 3 | 1 | $\left\lceil \frac{t}{3} \right\rceil$ |
| $R_2$ | 9 | 3 | $3\left\lceil \frac{t}{9} \right\rceil$ |
| $R_3$ | 4 | 1 | $\left\lceil \frac{t}{4} \right\rceil$ |

| Delay (ms) | $R_2$ |
|---|---|
| $h\left(\alpha_2, \beta_2^{[2,\,8]}\right)$ | 6 |
| $h\left(\alpha_2, \beta_2^{[4]}\right)$ | 5 |

**Table 1. First example**

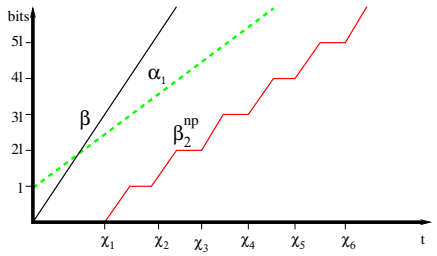


**Figure 5. Function $\beta_2^{np}(t)$**

with thanks to [2, Prop. 2], but the [2, Th. 5] uses the non-decreasing upper closure. We assume this is a typo and present here the corrected results.

[8] presents eq. (4), *i.e.* without the "self competitive" term, but does not explicitly say if this residual service is simple or strict. Implicitly, Real-Time Calculus uses strict services, and with this interpretation, the result of [8] is false, or there simply is a typo in the formula.

As an illustration, consider a server with a strict service curve $\beta(t) = t$ for three flows $R_1$, $R_2$ and $R_3$. Each flow $R_i$ is $\alpha_i$ upper-constrained, and has fixed packet size $l_i$, given in Table 1.

Figure 4 shows the residual service offered to flow $R_2$ using the results of [2, 8] ($\beta_2^{[2,\,8]}$ on the figure). With these results, the delay on the flow $R_2$ is 6 ms, as shown in Table 1. The curve $\beta_2^{[2,\,8]}$ shows that the remaining service increases from one unit, then two units subsequently. However, the flow of $R_2$ is expected to benefit from full server speed when it is served. In addition, $R_2$ has packets of size 3. As the priority is non preemptive, $R_2$ must send, without interruption, whole packages of size 3.

Briefly, these approaches only model the negative impact of non-preemption (the add of a negative term $-l_2^{\max}$) in the residual service, but not a positive one: when a non preemptive flow is served it benefits from the full speed from the server.

**Non-premption and residual service curve** The impact of non-preemption described above has been studied in [4], which gives an algorithm to compute a residual service with better delay bound than the one of [2, 8], assuming fixed size packets. The main idea of [4] is to consider non-preemption taking into account the effective size of the packets of lower priority. The curve in [2, 8] is then modified by an algorithm that increases the residual service curve each time a non-conforming growth is found. A non-conforming growth is detected whenever it is not a multiple of the packet size. The Figure 4 shows the residual service offered to $R_2$ using the results of [4] ($\beta_2^{[4]}$ on the figure). With these results, the delay on the flow $R_2$ is 5 ms, as shown in Table 1.

Although [4] improves the delay bound but does not clearly claim if the residual service is simple or strict[5]

### 2.3. Our contribution

We address the same problem as [4]. But, [4] leaves two open questions.

a) is there any analytical formulation of the result?
b) is the result tight?

Here we propose

a) an analytical expression of a residual *strict* service curve (with the proof);
b) an example showing the non-tightness of our result (last example in Section 5);
c) the generalisation of non-preemption with no assumption on the packet size of the high priority flow.

## 3. Context and modelling

We consider a server with a strict service curve $\beta$ for three flows $R_1$, $R_2$ and $R_3$, with a non preemptive strict priority policy, with $R_1$ the highest priority and $R_3$ the lowest. We also suppose that each flow $R_i$ is $\alpha_i$ upper-constrained. We also consider that $R_2$ and $R_3$ have fixed-size packets (of size $l_2$ and $l_3$ respectively). In the following, for every flow $R_i$, $R'_i$ represents its output. We also denote $R = \sum_i R_i$ and $R' = \sum_i R'_i$

### 3.1. Properties

Here are some important properties that will be used in the sequel.

The server has a strict service curve. It means for any backlogged period, $[t, s[$ of the aggregate flow $\sum_i R_i$.

$$\sum_{i=1}^{3} R'_i(s) - \sum_{i=1}^{3} R'_i(t) \geq \beta(s-t) \qquad (6)$$

### 3.2. Scheduling modelling

Here are presented some notions related to our modelling of scheduling and specially non preemptive priority scheduling.

When there is no backlog (*i.e.* $R'(t) = R(t)$), the server is efficient enough to serve the three flows simultaneously, and there is no influence of the scheduling policy.

On backlogged periods, we will consider that the three streams share the services of the server in mutual exclusion. Everything happens as if there was a scheduler responsible for allocating the server to the streams. That is to say, it exists a function $Sched : \mathbb{R} \mapsto \{1, 2, 3\}$, that, at each backlogged instant, gives the number of the flow served by the server. If $[t, s[$ is a backlogged period, $Sched([t, s[) = i$ means that the server is allocated to flow $i$ on $[t, s[$.

We call ***period of service*** $P$ for a flow $R$ a period of time during which the stream is served by the server *i.e.*

[5] In Section 5, it is shown that this service can not be strict (Table 3), and, as discussed in Section 2.1.4, it make a difference in some contexts.

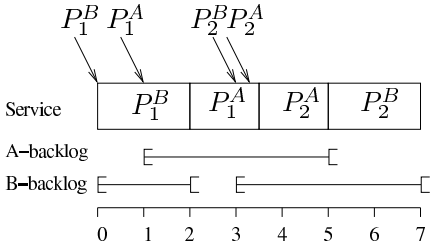


**Figure 6. Scheduling example**

$Sched(P) = i$. A *period of service* begins at a precise moment called ***start date of service***. We define $StSc$ the start date of service for a moment $t$. If $t$ is in a period of service $i$, *i.e.* $Sched(t) = i$ then $StSc(t)$ denotes the beginning of this period of service. Else, if $t$ is not in a period of service then $StSc(t)$ denotes the beginning of the next period of service. Then, $StSc(t)$ points either to the beginning of a service in the past or to some future point of time. We will consider a *period of service* as an open interval on the right and left-closed $[t, s[$. During the period of service of one flow, there is no output of the other flow.

These notions can be illustrated with two flows, $A$ and $B$, $A$ having higher priority than $B$, $A$ sending packets $P_1^A, P_2^A$, and $B$ sending packets $P_1^B, P_2^B$ (cf. Figure 6). The service times are represented in the Figure. Then $[1, 5[$ is a backlog period of $A$, but any subinterval also is ($[2, 4]$, $]2, 5[$, etc). The same, $[0, 2[$ and $[2, 7[$ are backlog periods of $B$, and $[0, 7[$ is a backlog period of $A + B$. The start of service can be also illustrated: $StSc_A(1) = StSc_A(3) = StSc_A(3.2) = 1$, $StSc_B(1) = 0$, $StSc_B(3) = StSc_B(6) = 5$.

This also illustrates the notion of "self competing term" presented in eq. 5: the $B$ backlog period $[3, 7[$ has to wait the service because the flow $A$ has been delayed by the packet $P_1^B$.

These definitions implies some obvious properties. Let $[t, s[$ be a backlogged period, for a flow $i$ then:

$$\forall u \in [t, s[: StSc_i(u) \geq StBl_i(u) \tag{7}$$

$$Sched([t, s[) = i \implies \forall j \neq i, \forall u, v \in [t, s[ : R'_j(u) = R'_j(v) \tag{8}$$

The behaviour of non preemptive strict priority is not so easy to model: consider Figure 7. The high priority flow $R_1$ has backlog from 0 to $u$, and emit data *at the server speed* up to $v$, and then stops. What is the decision of the scheduler at time $u$? There is no more $R_1$ backlog, so, it could switch to a low priority flow. Nevertheless, $R_1$ is still sending data. Since we are studying the low priority flow, we are making the pessimistic assumption that, at time $u$, the server is able to see that $R_1$ is still sending data, and still serves $R_1$ up to $v$. Then, the strict priority is modelled as follows: let $t$ be the start of a backlogged period. If there is input of the high priority flow, the server is allocated to this flow, *until its stop to emit*. Otherwise, the server is allocated to the low priority one, *until one packet has been transmitted* (*i.e.* until $s$ such that $R'_2(s) - R'_2(t) = l_2$). At the end of one period of service, the scheduling policy allocates the server as if it was the start of a backlogged period.

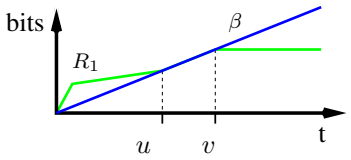


**Figure 7. Static priority modeling**

Therefore, we can write for three flows $i$, $j$ and $k$ $R'_i(s) - R'_i(t) \geq \beta(s-t) \wedge R'_j(s) = R'_j(t) \wedge R'_k(s) = R'_k(t)$ if $R_i$ is in a *period of service*.

By definition of start of service and backlog period:

$$R'(StSc(t)) = R'(StBl(t)) \tag{9}$$

$$\text{and then, } R'_i(StSc_i(t)) = R'_i(StBl_i(t)) \tag{10}$$

## 4. Contributions

Denote by $\psi_1$ an upper bound of the time necessary to $R_1$ to empty its queue, assuming that there was no accumulation time before. $\psi_1 = \inf\{t | \beta(t) - \alpha(t) > 0\} = (\beta - \alpha)^{str-1}(0)$. Denote by $\psi_i$ $(i > 1)$, an upper bound of time required to serve a packet of length $l_i$: $\psi_i = \inf\{t \mid \beta(t) \geq l_i\}$, $i > 1$. This means $\psi_i = \beta^{-1}(l_i)$. For a backlogged period $[t, s[$ of $R_2$, $x_i$ is the $i^{th}$ start of service of a packet of $R_2$ after $t$. That $i^{th}$ *period of service of packet* of $R_2$ during a backlogged period $[t, s[$ is an interval $[x_i, x_i + w_i[$, with $w_i = \inf\{t \mid R'_2(x_i + t) - R'_2(x_i) \geq l_2\}$. It means that

$$R'_2(x_i) - R'_2(x_1) = (i-1) \times l_2 \tag{11}$$

$$R'_2(x_i + w_i) - R'_2(x_1) = i \times l_2 \tag{12}$$

Between $t$ and $x_1$, $R_2$ may have received some service. However, since $t$ is arbitrary, $t$ is not necessarily the beginning of a service. Then,

$$0 \leq R'_2(x_1) - R'_2(t) < l_2 \tag{13}$$

The following is deduced from (11), (12) and (13).

$$R'_2(x_i) - R'_2(t) \geq (i-1) \times l_2 \tag{14}$$

$$R'_2(x_i + w_i) - R'_2(t) \geq i \times l_2 \tag{15}$$

### 4.1. Definitions related to the new residual service

The following term sets $\chi_i$, an upper bound of the period before the start date of the $i^{th}$ period of service of packet of $R_2$.

$\chi'_i = \inf\{t | \beta(t) - \alpha_1(t) > l_3 + (i-1)l_2\}$

$\chi"_i = \inf\{t | \beta(t + \psi_2) - \alpha_1(t + \psi_2) > i \times l_2\}$

and $\chi_i = \max\{\chi'_i, \chi"_i\}$.

With $f(t) = \beta(t) - \alpha_1(t)$ and $g(t) = f(t + \psi_2)$,

$\chi'_i = f^{str-1}(l_3 + (i-1) \times l_2)$ and $\chi"_i = g^{str-1}(i \times l_2)$.

We also define the following functions $x(t)$, $\chi(t)$ and $\beta_2^{np}$

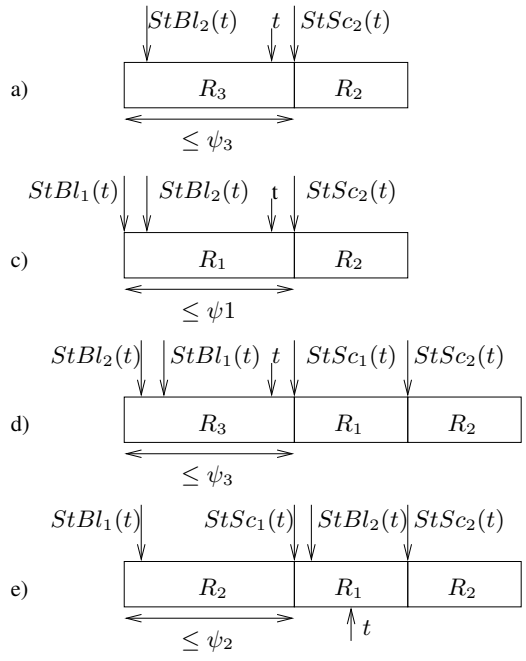


**Figure 8. Different cases for** $StSc_2(t)$

that will help us to present our results.
For $u \in [t, s[$, $x(u) = x_i : i = \max\{j : x_j \leq u\}$.
For $u \in [t, s[$, $\chi(u) = \chi_i : i = \max\{j : \chi_j \leq u\}$.

$$\begin{aligned}\beta_2^{np}(t) = \min\{&i \times l_2,\\ &\beta(s-t) - \beta(\chi'_i) + (i-1)l_2,\\ &\beta(s-t) - \beta(\chi"_i + \psi_2) + i \times l_2\} \quad (16)\end{aligned}$$

with $i$ such that $\chi(t) = \chi_i$. Figure 5 shows a trend of $\beta_2^{np}$.

### 4.2. Theorem

**Theorem 4.1.** *Consider a server that offers to three flows, $R_1$, $R_2$ and $R_3$, a strict service curve represented by a non-decreasing function $\beta$. Suppose that the flow $R_2$ (resp. $R_3$) emits its data into packets of fixed size $l_2$ (resp. of maximal size $l_3$). If the flow $R_1$ (resp. $R_2$) is $\alpha_1$ (resp. $\alpha_2$) upper-constrained and $R_1$ has a non preemptive priority over the flow $R_2$ and $R_2$ has a non preemptive priority over the flow $R_3$, then the server guarantees to $R_2$ a strict service curve $\beta_2^{np}(t)$ defined in eq (16).*

### 4.3. Proof of theorem

Let consider $[t, s[$ ,a backlogged period of $R_2$. Several cases exist.

$t$ does not belong to a backlogged period of $R_1$. Then, $R_2$ is the flow with the highest priority in the queue. Its transmission can be delayed by at most one packet of $R_3$. There are two sub-cases at time $StBl(t)$:

a) $R_3$ is served. $R_2$ can wait until the end of the transmission of a packet of $R_3$ (Figure 8-a).
b) $R_3$ is not served. $R_2$ does not suffer any delay.

$t$ belongs to a backlogged period of $R_1$. We can then distinguish three sub-cases at time $StBl(t)$.

c) $R_1$ is served. $R_2$ can wait for the transmission of $R_1$ until its queue is empty (Figure 8-c).
d) $R_3$ is served. In this case (Figure 8-d), $R_2$ wait for the transmission of one packet of $R_3$ and that $R_1$ empty its queue. Notice that, during the service time of $R_3$, $R_1$ can accumulate some backlog.
e) Another backlog period of $R_2$ is served. This is the equivalent notion of the "self competitive" term presented in the eq.5 and example of Figure 6 . Figure 8-e shows this case.

The first three cases (a-b-c) described above are special cases of the last two cases (d-e). We will focus only on cases that could lead to one of these two situations. We will denote by case 3-1-2 the situation where the flow $R_2$ waits for the end of the transmission of $R_1$ and $R_1$ have waited until the end of the transmission of a packet of $R_3$ (Figure 8-d). Similarly, we will denote by case 2-1-2 the situation where the flow $R_2$ waits for the end of the transmission of $R_1$ and $R_1$ have waited until the end of the transmission of a packet of $R_2$ (Figure 8-e). In the cases (d-e), $R_1$ begins to accumulate backlog at $StBl_1(t)$ when the flow $R_i$ ($i \in \{2, 3\}$) is being served. $StBl_1(t)$ belongs to a period of service of $R_i$. This period of service began at the moment $StSc_i(StBl_1(t))$. For the rest, let us denote $t_i^c = StSc_i(StBl_1(t))$, and note that

$$t_i^c \leq StBl_1(t) \quad (17)$$

The following results can now be presented before the proof.

**Lemma 4.2.** *case 2-1-2*
$StSc_1(t) - t_2^c \leq \psi_2$

*Proof.* One packet of $R_2$ is served between $t_2^c$ and $StSc_1(t)$.

$$\begin{aligned}l_2 &= R'_2(StSc_1(t)) - R'_2(t_2^c) \text{ (Only } R_2 \text{ is served)}\\ &\geq \beta(StSc_1(t) - t_2^c) \text{ (By definition of strict service)}\end{aligned}$$

Since $\beta$ is non-decreasing, then $StSc_1(t) - t_2^c \leq \beta^{-1}(l_2) = \psi_2$. □

**Lemma 4.3.** *case 3-1-2*
$StSc_1(t) - t_3^c \leq \psi_3$

*Proof.* Idem as Lemma 4.2. □

The next two results give an upper bound of waiting time before the beginning of the start of service for the $i^{th}$ packet of $R_2$

**Lemma 4.4.** *case 2-1-2:* $x_i - StSc_1(t) \leq \chi"_i$.

*Proof.* As in Lemma 4.2, we are going to use bounds on $R'_2(x_i) - R'_2(t_2^c)$ to deduce a bound on $x_i - t_2^c$. But since the definition of $\chi"_i$ is based on the pseudo-inverse of $\beta - \alpha$, which is not necessarily non-decreasing, we have to be more careful when manipulating $\inf$-based expression.

This is why a variable $u \in [0, s - t_2^c]$ is introduced in the proof.

By definition of this case, $R_3$ is not served in a backlogged period of $R_2$

$$R_3'(t_2^c + u) - R_3'(t_2^c) = 0 \tag{18}$$

For the flow $R_1$,

$R_1'(t_2^c + u) - R_1'(t_2^c)$
$= R_1'(t_2^c + u) - R_1'(StSc_1(t)) + R_1'(StSc_1(t)) - R_1'(t_2^c)$
$= R_1'(t_2^c + u) - R_1'(StSc_1(t))$
(since $R_1'(StSc_1(t)) = R_1'(t_2^c)$, from eq. 8)
$= R_1'(t_2^c + u) - R_1'(StBl_1(t))$ (from eq. 10)
$\leq R_1(t_2^c + u) - R_1(StBl_1(t))$ (because $R' \leq R$)
$\leq R_1(t_2^c + u) - R_1(t_2^c)$ (from Eq. 17)
$\leq \alpha_1(u)$ *i.e.*

$$R_1'(t_2^c + u) - R_1'(t_2^c) \leq \alpha_1(u) \tag{19}$$

By definition of the strict service curve,

$$R'(t_2^c + u) - R'(t_2^c) \geq \beta(u) \tag{20}$$

Combining 20, eq. 19 and 18, we get

$$R_2'(t_2^c + u) - R_2'(t_2^c) \geq \beta(u) - \alpha(u) \text{ then}$$
$$\inf\{u | R_2'(t_2^c + u) - R_2'(t_2^c) > i \times l_2\} \leq$$
$$\inf\{u | \beta(u) - \alpha(u) > i \times l_2\} = \chi"_i + \psi_2$$

And because $u \mapsto R_2'(t_2^c + u) - R_2'(t_2^c)$ is non-decreasing: $x_i - t_2^c \leq \chi"_i + \psi_2$. Then, from Lemma 4.2 $x_i - StSc_1(t) \leq \chi"_i$. □

**Lemma 4.5.** *case 3-1-2:* $x_i - t_3^c \leq \chi_i'$

*Proof.* Idem as Lemma 4.4. □

*Proof of Theorem 4.1.* Consider $[t, s[$, a backlogged period of $R_2$. Let $i = \max\{j \in \mathbb{N} | x_j \leq s\}$.

If $s \geq x_i + w_i$: $s$ is not in a period of service of $R_2$
$R_2'(s) - R_2'(t) \geq i \times l_2$ (by definition of $x_i$)
If $x_i \leq s < x_i + w_i$: $s$ is in a period of service of $R_2$ ($R_1'(s) = R_1'(x_i)$ and $R_3'(s) = R_3'(x_i)$). Then
$\begin{cases} R_1'(s) - R_1'(t) = R_1'(x_i) - R_1'(t) \\ R_3'(s) - R_3'(t) = R_3'(x_i) - R_3'(t) \end{cases}$
then, $R_2'(s) - R_2'(t) \geq \beta(s-t) - (R_1'(x_i) - R_1'(t) + R_3'(x_i) - R_3'(t))$

- case 2-1-2

$R_1'(x_i) - R_1'(t) \leq \alpha_1(x_i - t_2^c)$ (from Eq. 19)
$\leq \alpha_1(x_i - StSc_1(t) - \psi_2)$ (from Lemma 4.2)
$\leq \alpha_1(\chi"_i + \psi_2)$ (from Lemma 4.4). Then, we have:

$\begin{cases} R_1'(x_i) - R_1'(t) \leq \alpha_1(\chi"_i + \psi_2) \\ R_3'(x_i) - R_3'(t) = 0 \end{cases}$
then, $R_2'(s) - R_2'(t) \geq \beta(s-t) - \alpha_1(\chi"_i + \psi_2)$.
By definition of $\chi"_i$, $\beta(\chi"_i + \psi_2) - \alpha_1(\chi"_i + \psi_2) > i \times l_2$.
Then, $R_2'(s) - R_2'(t) \geq \beta(s-t) - \beta(\chi"_i + \psi_2) + i \times l_2$

| | | | |
|---|---|---|---|
| $\alpha_1(t)$ | $\begin{cases} \frac{1}{2}t + 3 \text{ if } t \geq 0 \\ 0 \text{ else} \end{cases}$ | $h\left(\alpha_2, \beta_2^{\text{spl}}\right)$ | 10 |
| $\alpha_2(t)$ | $2 \times \left\lceil \frac{t}{4} \right\rceil$ | $h\left(\alpha_2, \beta_2^{\text{np}}\right)$ | 10 |
| | | $h\left(\alpha_2, \beta_2^{[2,\,8,\,4]}\right)$ | 14 |

**Table 2. Second example (with two flows)**

- case 3-1-2

$R_1'(x_i) - R_1'(t) \leq \alpha_1(x_i - t_3^c)$
$\leq \alpha_1(\chi_i')$ (from Lemma 4.5). Then, we have:

$\begin{cases} R_1'(x_i) - R_1'(t) \leq \alpha_1(\chi_i') \\ R_3'(x_i) - R_3'(t) \leq l_3 \end{cases}$
then, $R_2'(s) - R_2'(t) \geq \beta(s-t) - \alpha_1(\chi_i') - l_3$.
By definition of $\chi_i'$, $\beta(\chi_i') - \alpha_1(\chi_i') > l_3 + (i-1)l_2$.
Then, $R_2'(s) - R_2'(t) \geq \beta(s-t) - \beta(\chi_i') + (i-1)l_2$

Finally,

$$R_2'(s) - R_2'(t) \geq \min\{i \times l_2, \beta(s-t) - \beta(\chi_i') + (i-1) \times l_2, \beta(s-t) - \beta(\chi"_i + \psi_2) + i \times l_2\}$$

□

**Corollary 4.6** (Generalisation to any number of flows)**.** *The result with 3 flows can be generalised to any number of flow $A_1, \ldots, A_n$. Let us consider the flow $A_i$, and make the renaming $R_2 = A_i$, $R_1 = \sum_{j=1}^{i-1} A_j$, $R_3 = \sum_{j=i+1}^{n} A_j$. The sum of higher priority flow can be seen as a unique flow, whose arrival curve is the sum of the individual arrival curves, and the same for the low priority ones* [6].

## 5. Application

As in Section 2.2, we are going to compare the approaches of [2, 8, 4] and the one presented here on a simple example where a server with a strict service curve $\beta$ is shared by two flows ($R_1$, $R_2$), or three flows ($R_1$, $R_2$, $R_3$).

A first example with three flows is the one already presented in Section 2.2 (cf. Table 1 and Figure 4). In this case, the curve $\beta_2^{np}$ is exactly the same than the one of [4]. In this case, our contribution is to give an analytical solution to the problem ([4] gives an algorithm, not an expression), with stronger properties (strict service vs simple service).

A second example with two flows is the one with $\beta(t) = t$, $l_2 = 2$, $\alpha_1$ and $\alpha_2$ depicted on Table 2 . In this example, we do not assume fixed packet size for the high priority flow: the packet size is unknown. In this case, a fluid arrival curve for the high priority flow must be considered.

In this case, the improvements from [4] can not be applied, and the delays are equivalents with the one of [2, 8] (the server guaranties for $R_2$ a simple service curve $\beta_2^{\text{spl}} =$

[6] In case of $A_i = A_n$, just consider $R_3 = \alpha_3 = l_3 = 0$.

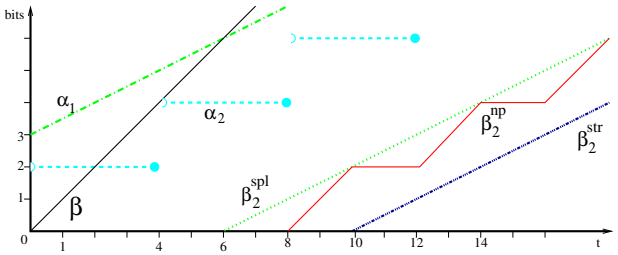


**Figure 9. Curves comparisons (Table 2)**

| | Period | Size ($l_i$) | $\alpha_i$ | Delay, [7] | Delay, [4] | Our delay |
|---|---|---|---|---|---|---|
| $R_1$ | 2.5 | 2.5 | $2.5\left\lceil\frac{2t}{5}\right\rceil$ | 2 | 2 | 2 |
| $R_2$ | 3.5 | 2.5 | $2.5\left\lceil\frac{2t}{7}\right\rceil$ | 3 | 3 | 3 |
| $R_3$ | 3.5 | 2.5 | $2.5\left\lceil\frac{2t}{7}\right\rceil$ | 3.5 | 3.5 | 6 |

**Table 3. Third example: CAN**

$[\frac{1}{2}t-3]^+$ [2] and a strict service curve $\beta_2^{\text{str}} = [\frac{1}{2}t-5]^+$ [2, 8, 4]).

On the contrary, our approach is still able to identify instants where the low priority flow will benefit from the full server speed. The new service curve $\beta_2^{\text{np}}$ is between the two others. The delays ($h(\alpha_2, \beta_2^{\text{spl}})$, $h(\alpha_2, \beta_2^{\text{np}})$, $h(\alpha_2, \beta_2^{[2,\,8,\,4]})$) are summarized in the Table 2.

The last example with three flows compares our approach with the famous CAN scheduling example presented in [7] (cf Table 3, with $\beta(t) = 2.5t$). In this example, it is known that [7] gives the exact worst case. Our approach does not give the exact worst case, but is still sound.

This example can also be used to verify that the service provided in [4] is neither strict nor weakly strict. This example is the same as the one presented in [4, Fig. 5], up to a 50% scale. Let us consider a specific scheduling: all tasks start at date 0, and are strictly periodic (see Figure 10). Let now consider $s = 3.5$, $t = 6$ ($[s,t]$ is a backlogged period for C, $s = StBl_C(t)$, $t - s = 2.5$). If $\beta_C^{[4]}$ was the curve of a strict or weakly strict service, one should have $R'_C(t) - R'_C(s) \geq \beta_C^{[4]}(t-s)$, but $\beta_C^{[4]}(t-s) = 0.5$, and $R'_C(t) - R'_C(s) = 0$ (no $C$ output on $[s,t]$).

## 6. Conclusions

Static priority is a simple and widely used scheduling policy. It has been studied in network calculus, and the first results, [1], had some technical limitations. Further works have passed these limitations [2, 8], but they only model the negative impact of non-preemption (the high priority flow can be interrupted by one low priority packet), not its benefits (when the low priority flow is served, it has the full server speed, even if from the long term point of view, it only has a residual part). This aspect is taken into account by [4], which gives an algorithm to improve the residual curve of [8].

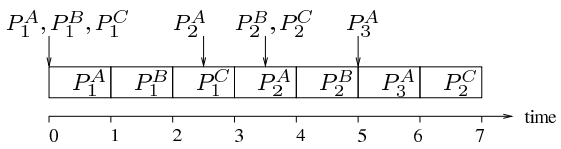


**Figure 10. Example scheduling with CAN**

In this paper, this aspect is also considered, but an analytical expression is given for a *strict* residual service curve (with its proof), with less hypotheses (the high priority flow can have non-fixed packet size). Compared to other theories, our approach is more general, but does not compute the exact worst case. This means that our network calculus-based approach is not tight: it is more general than [7] (no assuming of periodic flow, neither a constant service) but gives pessimistic results in some cases.

These are very interesting issues: first, it would be interesting to generalise the approach to the case of variable-size packets in the low priority flows, and finally, it would also be interesting to see why we do not get the exact worst case.

## References


[1] J.-Y. L. Boudec and P. Thiran. *Network Calculus*, volume 2050. Springer verlag - Lecture Notes in Computer Sciences, 2001.

[2] A. Bouillard, L. Jouhet, and E. Thierry. Service curves in network calculus: dos and don'ts. Rapport de recherche INRIA 7094, INRIA, November 2009.

[3] C. Cheng-Shang. *Performance Guarantees in Communication Networks*, volume ISBN : 1-85233-226-3. Springer-Verlag, 2000.

[4] D. B. Chokshi and P. Bhaduri. Modeling fixed priority non-preemptive scheduling with real-time calculus. In *Proc. of the 2008 14th IEEE int. conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA'08)*, Kaohsiung, Taiwan, 2008.

[5] R. L. Cruz. A calculus for network delay, part I: Network elements in isolation. *IEEE Transaction on Information Theory*, 37(1):114–131, 1991.

[6] R. L. Cruz. A calculus for network delay, part II: Network analysis. *IEEE Transaction on Information Theory*, 37(1):132–141, 1991.

[7] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised, 2007.

[8] W. Haid and L. Thiele. Complex task activation schemes in system level performance analysis. In *ESWeek '07: Proc. of the 5th IEEE/ACM int. conf. on Hardware/Software Codesign and System Synthesis*, pages 173–178, Salzburg, Austria, September 30 - October 03, 2007.

[9] S. Perathoner, E. Wandeler, L. Thiele, A. Hamann, S. Schliecker, R. Henia, R. Racu, R. Ernst, and M. Gonzlez Harbour. Influence of different abstractions on the performance analysis of distributed hard real-time systems. *Design Automation for Embedded Systems*, 13:27–49, 2009. 10.1007/s10617-008-9015-1.