Deficit Round Robin with Network Calculus

Marc Boyer
ONERA – The French Aerospace Lab
F 31055 , Toulouse, France
Marc.Boyer@onera.fr

Giovanni Stea
Dep. of Information Engineering
University of Pisa
Largo L. Lazzarino 2, Pisa, Italy
g.stea@iet.unipi.it

William Mangoua Sofack
ONERA – The French Aerospace Lab
F 31055 , Toulouse, France
William.Mangoua_Sofack@onera.fr

***Abstract*—[1] Generalised Processor Sharing (GPS) is a well-known ideal service policy designed to share the capacity of a server among the input flows fairly: each backlogged flow receives a pre-defined fraction of the total server capacity, according to its *weight*. Several practical implementations of GPS have been proposed, among which Deficit Round Robin (DRR) is widely deployed since it can be implemented in a very efficient way. The worst-case performance of DRR has been studied by several papers, all of which assume that the shared server has a constant rate. This paper studies DRR using Network Calculus, under very general assumptions. Latency results that generalise all the previous works are derived, and a residual service is derived from DRR parameters. This residual service is shown to be as good as or even better than previous studies when restricting it to the same assumptions.**

I. Introduction

Critical real-time systems are commonly made of several real-time processing units exchanging information through a network. Then, the global correctness of the systems relies not only on the real-time performance of the processing units, but also on the capacity to bound the delay introduced by the network (aka Worst Case Traversal Time – WCTT). Since for cost reasons (e.g., energy, weight, maintenance) this network is often shared by several functions and/or several flows, methods allowing to handle shared networks must be used to compute WCTT bounds.

Network calculus [7] is such a theory, with a strong mathematical background, and some success stories: it has been used to certify the A380 AFDX backbone [4].

One common criticism made to network calculus is its pessimism: it computes upper bounds on the delay, not the exact worst-case delay. In fact, it has been shown that the pessimism of the network calculus on common AFDX configuration was not so high as people thought: about 16.5% [3]. Moreover, people often restrict network calculus to (σ, ρ)-calculus, *i.e.* a calculus were server only offer rate-latency capacity ($\beta_{R,T}$ service curves in network calculus), and flows are constrained by token buckets ($\gamma_{r,b}$ arrival curves – see Section II for details). However the theory allows other types of curves, offering a better modelling of flows and then more realistic bounds.

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

In particular, a recent work [14] has shown that network calculus is both more general and as good as existing scheduling-based analyses for the non-preemptive static priority policy (NP-SP).

In this work, the Deficit Round Robin policy is analysed through Network Calculus. The main contribution is a theorem that allows per-flow service guarantees to be derived from an aggregate service curve, provided that the latter is *strict* and that flows share the system capacity using DRR. The above modelling generalises all previous works, [16], [6], [9], where latency modelling is done under more restrictive hypotheses (chiefly, constant-rate server), and our results are also as good as the existing ones under the same hypotheses.

After a first presentation of Network Calculus in Section II, Section III presents the Deficit Round Robin algorithm , its network calculus modelling and the residual service offered to each flow. Section IV presents previous works, and compares the accuracy of the different solutions. Section VI concludes the paper.

II. Network Calculus

Network Calculus analysis focuses on worst-case performance in networks. 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, in order to compute bounds on buffers size or delays.

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

VALUETOOLS 2012, October 09-12, Cargèse, France

DOI 10.4108/valuetools.2012.250202

- **Flooring $\lfloor\cdot\rfloor$ and ceiling $\lceil\cdot\rceil$:** $\forall x \in \mathbb{R}$
  $\lfloor x\rfloor \in \mathbb{N}$, $\lfloor x\rfloor \leq x < \lfloor x\rfloor + 1$
  $\lceil x\rceil \in \mathbb{N}$, $\lceil x\rceil - 1 < x \leq \lceil x\rceil$
- **The Vertical deviation:** It is defined for two functions f and g by $v(f,g) = \sup_{t\geq 0}\{f(t) - g(t)\}$
- **The Horizontal deviation:** It is defined for two functions f and g by $h(f,g) = \sup_{t\geq 0}\{\inf\{d \geq 0 \mid f(t) \leq g(t+d)\}\}$
- **The Min-plus convolution:** It is defined for two functions f and g by $(f*g)(t) = \inf_{0\leq s\leq t}\{f(t-s) + g(s)\}$
- **The Positive and non-decreasing upper closure:** It is defined for a function f by $f\uparrow(t) = [\sup_{0\leq s\leq t} f(s)]^+$

To model flows constraint and service guarantees, network calculus uses a set of usual parametrised curves, δ_d, λ_R, $\beta_{R,T}$, $\gamma_{r,b}$, defined by:

$$\delta_d(t) = \begin{cases} 0 \text{ if } t \leq d \\ \infty \text{ otherwise} \end{cases} \qquad \gamma_{r,b}(t) = \begin{cases} rt + b \text{ if } t > 0 \\ 0 \text{ otherwise} \end{cases}$$

$$\lambda_R(t) = Rt \qquad \beta_{R,T}(t) = R[t-T]^+$$

$$\nu_{T,\tau}(t) = \min\left(\delta_0, \left\lceil \frac{t+\tau}{T} \right\rceil\right)$$

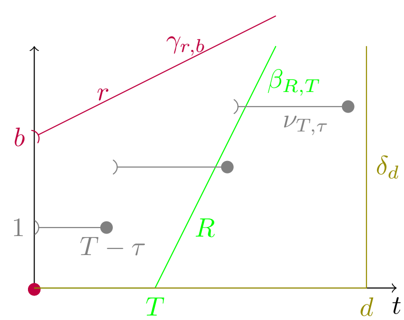


Fig. 1. Common curves

B. Network calculus: reality modelling

A network calculus model for a communication network consists in the three following components:

1) 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).
2) 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.
3) A description of the behaviour of each subsystem, that is service curves bounding the performance 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, *flows* are modelled by cumulative functions $R \in \mathcal{F}$: $R(t)$ counts the total amount of data generated by the flow up to time t.

The *servers* are just relations between some input and output flow ($S \in \mathcal{F} \times \mathcal{F}$). Then $(R, R') \in S$, denoted $R \xrightarrow{S} R'$, means that a server S receives an input flow, $R(t)$, and delivers the data after a variable delay. We have inequality $R' \leq R$, 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.

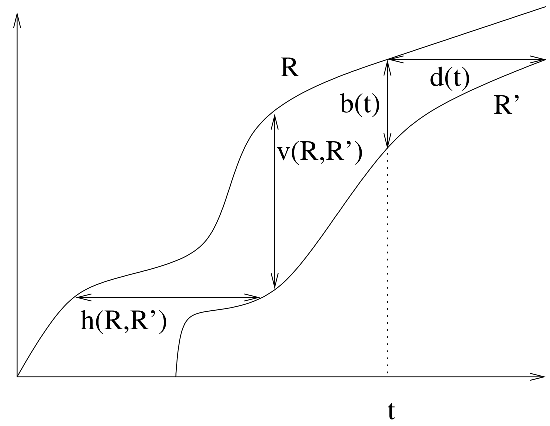


Fig. 2. Backlog and delay

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 [7]. 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 an interval during which the backlog is non null. If t is an instant within a backlogged period, the backlogged period has started at $StBl(t)$ defined by:

$$StBl(t) = \sup\{u \leq t \mid R'(u) = R(u)\} \tag{1}$$

A *maximal backlog period* is an interval I such that if it exists a backlog period I' such that $I \subseteq I'$, then, $I = I'$.

The *virtual delay*[2] at a time t, $d(t)$, is the delay that a bit entered at time t will wait before exiting the system, and it is defined as:

$$d(t) = \inf\{\tau \geq 0 \mid R(t) \leq R'(t+\tau)\} \tag{2}$$

Obviously $d(t) \leq h(R, R')$.

These notions are illustrated in Figure 2.

C. Network calculus: contract modelling

To provide guarantees for data flows, some traffic contracts on the traffics and the services in the network are needed. For this purpose, network calculus provides the concepts of arrival curve and service curve.

[2]Some authors makes the distinction that this definition implies a FIFO scheduling. The point of view in this paper is that, when considering *a single flow*, this is the only reasonable definition. Other definitions of delay imply to be able to distinguish different kinds of bits in a flows, *i.e.* to have different aggregated flows, as will be presented in Section II-D

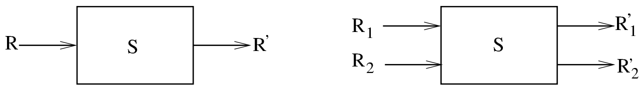


Fig. 3. Servers.

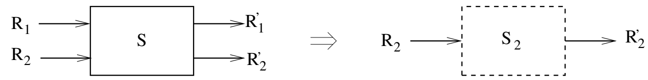


Fig. 4. Residual server and residual service

a) 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 also say that R *has* α *as an arrival curve/* This condition is equivalent to $R \leq R * \alpha$.

Function $\gamma_{r,b}$ models the token-bucket contract: the flow can send a burst of size b, and has a long-term rate r. Function $s\nu_{T,\tau}$ models a sporadic flow with packets of maximal size s, a pseudo-period T and a jitter τ.

b) Service curve: The behaviour of a server is modelled through the concept of service curve, modelling some guarantees on the service provided to flows.

The literature offers several definitions for different types of service guarantees. [1] proposes a comparative study of service guarantees. Consider a system $R \xrightarrow{S} R'$, *i.e.* a server S with input R and output R' (Figure 3).

Server S offers a *simple* (or *weak*) service curve β if and only if, for all pair $R \xrightarrow{S} R$, $R' \geq R * \beta$.

We say that a system S offers a *strict* service curve β if, for all pair $R \xrightarrow{S} R'$, during any backlogged period $)t, s]$, we have $R'(s) - R'(t) \geq \beta(s-t)$.

There is a hierarchy between these service notions. A strict service curve is also a simple service curve. As discussed in Section II-D, the two definitions exhibit different with respect to decomposition of residual service.

Let us now present the main network calculus results:

Theorem 1 (Backlog and delay bound): Assume a flow, constrained by an arrival curve α, traverses a system that offers a service curve β:

The backlog $b(t)$ for all t satisfies: $b(t) \leq v(\alpha, \beta)$.

The virtual delay $d(t)$ for all t satisfies: $d(t) \leq h(\alpha, \beta)$.

D. Aggregation and residual service

In general, servers do not provide service to a single flow, rather they share their capacity among a set of flows. The server definition has to be generalised to multiple-input / multiple-output servers: $S \in \mathcal{F}^n \times \mathcal{F}^n$, $(R_1, \ldots, R_n) \xrightarrow{S} (R'_1, \ldots, R'_n)$. And the capacity of the server is shared by several flows.

The projection on the i-th flow defines the *residual server* S_i: $R_i \xrightarrow{S_i} R'_i$. The notions of delay, backlog and backlogged period can the be defined per flow, considering the residual server. Server S is "R_i backlogged" at time t iff $R'_i(t) > R_i(t)$.

Modelling aggregation and residual service 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 β to two flows R_1 and R_2 , 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 repartition of the service among the flows depends on the server policy (common policies are FIFO [10], static priorities [5], [1], P-GPS/WFQ [15], DRR [16]).

The global idea, in network calculus, is to consider the residual server S_i, and to derive a *residual service* (simple or strict) of curve β_i offered by this server.

An important issue is the tightness: in network calculus, a residual service is said to be *tight* if it allows one to compute *the worst-case delay*, i.e. an attainable upper bound, rather than a pessimistic, overrated upper bound on the delay.

When dealing with aggregation and residual service, the type of service curve plays a crucial role: some results require that the service curve be strict, others may be obtained assuming simple service curves[3]. The residual service curve can itself be simple or strict. This is of importance in the case of more than two flows, since a residual service curve can itself be shared by aggregated flows.

III. Deficit Round Robin

Different policies have been defined to share the capacity of a server among several flows. GPS is an ideal policy explicitly designed to allow fair sharing. In GPS, each flow R_i receives a fractional part $\rho_i \beta$ of the system service curve β (with $\sum_i \rho_i = 1$). As mentioned in [15]: "a problem with GPS is that it is an idealised discipline that does not transmit packets as entities. It assumes that the server can serve multiple sessions simultaneously and that the traffic is infinitely divisible". A well-known GPS approximation is P-GPS (aka Weighted Fair Queueing [15]), that offers the same guarantees as GPS in a packet system, up to one packet size deviation. Nevertheless, P-GPS implementation is rather complex, and other GPS approximations are used. The Deficit Round Robin algorithm is a practical, efficient implementation of the GPS paradigm. In DRR, each flow R_i receives a credit Q_i, and DRR ensures that each flow will get a factional part $\rho_i = \frac{Q_i}{\sum_{j=1}^{n} Q_j}$ of the service. DRR is among the most used, since it exhibits a low complexity ($O(1)$) and can be implemented in very efficient ways (like the Aliquem implementation of [8]). Nevertheless, DRR latency is larger than the P-GPS one. Then, this latency must be carefully evaluated.

Papers about GPS consider that the shared server has a constant rate service ($\beta = \lambda_r$), but the definition can be generalised to any kind of β service curve in the context of network calculus: the generalisation of GPS just states that the aggregated service β is shared in a way such that each flows receives a service curve $\rho_i \beta$.

Similarly, DRR makes no assumption about the kind of curve offered by the server. However, all the performance

[3]In fact, up to now, most results need a *strict* service curve, and the FIFO policy is the only one requiring only simple service curve.

analyses in the literature are carried out under the assumption of a constant rate service.

A. DRR algorithm

The DRR algorithm is explained in Algorithm 1. We assume that there are n input flows, and when a packet of i-th flow enters the system, it is put into the i-th queue. The behaviour of the scheduler is the following: an infinite global loop scans all queues in sequence. If the i-th queue is non empty, it is selected, the *Deficit Counter* $\texttt{DC}[i]$ is increased by Q_i, called the flow's *quantum*, and as long as $\texttt{DC}[i]$ is greater than or equal to the size of the head-of-line packet, the packet is sent and the $\texttt{DC}[i]$ counter is decreased accordingly. The loop ends when the flow lacks enough credit to send the next packet or its queue gets empty. In this last case, $\texttt{DC}[i]$ is set to zero.

A minor modification has been made with respect to the original presentation of [16]: the infinite loop scans all queues, whereas the original code maintains a list of *active* (*i.e.*, backlogged) queues and scans only these. It is assumed here that DRR implementation cost is negligible.

To increase the readability of the proof (given in the Appendix), a `print` pseudo-instruction has been added, executed in null time, which prints the current time-stamp `now()`, and the flow i which is currently selected. Of course, this instruction is not part of a real implementation.

Note that no assumption is done about the policy *inside* each queue. The only restriction is that the head of queue is the same, from line 9 to line 11 in a given iteration of the loop. But it can be changed from one iteration to the other, if, for example, a packet with a higher priority arrives between two iterations of the service.

Input: Per flow quantum: $Q_1..Q_n$ (Integer)
Data: Per flow deficit: $\texttt{DC}$[1..n] (Integer)
Data: Counter: k (Integer)
1 **for** *i= 1 to n* **do**
2     $\texttt{DC}[i] \leftarrow 0$;
3 **end**
4 **while** *True* **do**
5   **for** *i= 1 to n* **do**
6   **if** *not empty(i)* **then**
            `// Print is a pseudo instruction, used to simplify the proof`
7   `print` (now(), i) ;
8             $\texttt{DC}[i] \leftarrow \texttt{DC}[i] + Q_i$;
9   **while** *(not empty(i)) and (size(head(i))* $\leq$ *DC[i])* **do**
10   `send`(head(i)) ;
11                 $\texttt{DC}[i] \leftarrow \texttt{DC}[i] -$ size(head(i)) ;
12   removeHead(i) ;
13   **end**
14   **if** *empty(i)* **then**
15                 $\texttt{DC}[i] \leftarrow 0$
16   **end**
17   **end**
18   **end**
19 **end**

Algorithm 1: DRR algorithm

B. DRR modelling

We are going to make a few assumption in the proof: mainly, we assume that

- only the `send` instruction takes time,
- the `send` instruction always has non null duration,
- the same queue can be not seen as empty and non empty at the same date, *i.e.* if a queue i is seen as empty at line 14, then, it can not be seen non empty at the next iteration at line 9 if this iteration occurs at the same instant.

C. DRR residual service

We now state the main result.

Theorem 2 (Residual DRR service): Let S be a server, offering a strict service of super-additive curve β, shared by n flows $R_1, \dots, R_n$. Each flow has a maximum packet size l_i^m, and a quantum Q_i. Then, S offers to each flow R_i a strict residual service curve β_i^{DRR} defined by:

$$\beta_i^{DRR}(t) = \left[\frac{Q_i}{F}\beta(t) - \frac{Q_i(L - l_i^m) + (F - Q_i)(Q_i + l_i^m)}{F} \right]^+$$

with $F = \sum_{i=1}^n Q_i$, $L = \sum_{i=1}^n l_i^m$.

Moreover, if the packet length is discrete, multiple of a basic unit ϵ (e.g., one byte), and all the Q_i are multiple of this basic unit as well, then the residual service curve can be refined as:

$$\beta_i^{\epsilon\text{-}DRR}(t) = \left[\frac{Q_i}{F}\beta(t) - \frac{Q_i(L^\epsilon - l_i^{m\text{-}\epsilon}) + (F - Q_i)(Q_i + l_i^{m\text{-}\epsilon})}{F} \right]^+$$

with $l_i^{m\text{-}\epsilon} = l_i^m - \epsilon$ and $L^\epsilon = \sum_{i=1}^n l_i^{m\text{-}\epsilon}$.

Such a basic unit always exists in practice, its value being at least $\epsilon = 1$ or $\epsilon = 8$, due to sizes being in bits or bytes. Nonetheless, it is sometimes easier from a modelling point of view to deal with continuous packet lengths. This is why both curves are presented.

c) Sketch of proof: A full proof is reported in the Appendix. The proof is quite simple. The first two observations are that the value of $\texttt{DC}[i]$ always lies within an interval $[0, l_i^m + Q_i)$ (and, in fact, within $[0, l_i^m)$ when considering values at line 6), and increases of at most Q_i between two iterations of the main loop. Then, during p consecutive iterations, a flow i can send at most $pQ_i + l_i^m$ data. If a flow i is continuously backlogged during an interval I and has p service opportunities during I, then, it can send at least $pQ_i - l_i^m$ data.

Now, consider two instants of a backlogged period of a flow i, s and t, $s \leq t$. Let p denote the number of service opportunities for i between s and t. Then, each flow $j \neq i$ gets at most $p+1$ service opportunities between s and t.

From the strict service curve property, we get $\beta(s,t) \leq R_i'(s,t) + \sum_{j \neq i} R_j'(s,t)$. Let p be the number of full service opportunities of i between s and t. Then, $\sum_{j\neq i} R'_j \leq \sum_{j\neq i}((p+1)Q_j + l^m_j) = (p+1)\sum_{j\neq i} Q_j + \sum_{j\neq i} l^m_j$, which yields a lower bound for p. It is also known that $R'_i(s,t) \geq pQ_i - l^m_i$. By substituting the lower bound for p in the above formula, the thesis follows after some straightforward manipulations.

The only subtle points are the formal definitions of "between", "period of service" and so on.

d) Types of services: In Theorem 2, the service offered by the shared server must be strict, and the residual service offered to each flow is still strict. Strictness is an important property: in general, real implementations offer strict service curves. However, the challenge is to get strict *residual* service curves from the former, since this allows one to apply residuation again in order to get performance guarantees for flows in a hierarchical scheduling context. For instance, a single DRR queue could be shared among several sub-flows according to some policy. In this case, since each queue is given a strict service curve, performance guarantees for the sub-flows can be derived through residuation.

For example, since the DRR latency may be too high for some flows, commercial routers allow a NP-SP/DRR policy, meaning that flows are aggregated into classes of service, with a first level of static priority scheduling (NP-SP), and, inside each priority class, the flows can be scheduled with a DRR policy. Then, in order to be able to apply our result, the residual service curve of the NP-SP policy must be strict.

e) Parameter choices: Some papers [16], [6] consider only a single maximum packet size, which is assumed to be common to all flows. However, considering per-flow maximal packet sizes instead appears to be a reasonable assumption (flows with small packets – e.g., VoIP flows – normally coexist with flows with large packets – e.g., web flows), and yields tighter bounds [11], [9], [8]. The basic unit ϵ has been introduced to ensure comparison with [6]. If the packet length and quanta are discrete, the interval within $DC[i]$ always falls, which were reported to be $[0, l^m_i + Q_i)$ and $[0, l^m_i)$, can indeed be rewritten as $[0, l^m_i - \epsilon + Q_i]$ and $[0, l^m_i - \epsilon]$. This allows one to save an ϵ worth of latency for each flow wherever an l^m_i term appears. The expected gain of modelling quanta and packets as discrete entities (rather than continuous) is however negligible in practical cases, where ϵ is one byte and l^m_i is commonly hundreds or thousands of bytes. For a 100-byte packet (quite small nowadays), the improvement represents one percent.

IV. Related works

DRR has been designed in [16] for fair sharing of server capacity among flows, *i.e.* as a possible implementation of the ideal GPS policy [15], with a low implementation complexity ($O(1)$ if each quantum Q_i is no less than the maximal packet size [16, Th. 4.5]). Its drawback is its latency. For flow i, latency includes a quantum plus a max packet size from all flows $j \neq i$, hence it grows as $O(n)$. Furthermore, latency is not proportional to a flow's quantum: increasing a flow's bandwidth does not decrease its latency. This makes it hard to meet tight delay constraints when many flows contend for bandwidth at a server.

A. DRR latency evaluations

The latency of the DRR is defined as a time deviation with the ideal GPS policy. Several works have dealt with evaluating this latency [17], [6], [9]. However, all papers consider that the server shared according to the DRR policy is a constant rate server (λ_r in network calculus), and see the residual service for a flow as a Latency-rate (or rate-latency) server [18], ($\beta_{R,T}$ in network calculus). In this framework, all papers show that the residual service for flow i is a $\beta_{\frac{Q_i}{F}r,\theta_i}$, and each paper provides its own evaluation of θ_i.

The first work, in [17], gives the bound $\theta^{[17]}_i$ on latency:

$$\theta^{[17]}_i = \frac{3F - 2Q_i}{r} \tag{3}$$

This bound has been refined in [6]. The notations used in [6] are a little bit different (using a W value equal to $F/\min_j Q_j$, $w_i = Q_i/\min_j Q_j$). Here we present the result with the current notations:

$$\theta^{[6]}_i = \frac{1}{r}\left(F - Q_i + (l^m - 1)\left(\frac{F}{Q_i} + n - 2\right)\right) \tag{4}$$

with $l^m = \max_j \left\{l^m_j\right\}$.

Independently, a third bound was found in [8], [9]:

$$\theta^{[9]}_i = \frac{1}{r}\left((F - Q_i)(1 + \frac{l^m_i}{Q_i}) + \sum_{j=1}^{n} l^m_j\right) \tag{5}$$

Last, [19] considers an AFDX network with a DRR scheduling, using network calculus. But it also restricts the study to constant rate server, and computes a bound on latency with this assumption. Moreover, since the focus in [19] is done on comparing DRR, FIFO and NP-SP policies, technical details allowing an exact comparison are missing. Its latency seems equivalent to the one of [8].

In order to compare our result with the other ones, we must restrict to the case $\beta(t) = \lambda_r(t) = rt$, even though the result of Theorem 2 is more general. Considering that $\beta = \lambda_r$, our result gives:

$$\theta_i = \frac{1}{r}\left((L^\epsilon - l^{m\text{-}\epsilon}_i) + (F - Q_i)\left(1 + \frac{l^{m\text{-}\epsilon}_i}{Q_i}\right)\right) \tag{6}$$

Proof: The above result can be proved through straightforward algebraic manipulations:

$$\begin{aligned}\beta_i(t) &= \left[\frac{Q_i}{F}rt - \frac{Q_i(L^\epsilon - l^{m\text{-}\epsilon}_i) + (F - Q_i)(Q_i + l^{m\text{-}\epsilon}_i)}{F}\right]^+ \\ &= \frac{Q_i}{F}r\left[t - \frac{1}{r}\left((L^\epsilon - l^{m\text{-}\epsilon}_i) + (F - Q_i)\frac{Q_i + l^{m\text{-}\epsilon}_i}{Q_i}\right)\right]^+ \\ &= \beta_{\frac{Q_i}{F}r,\theta_i}\end{aligned}$$

■

We now show that, under the same assumptions, our result is equal to or better than the two others [6], [9].

To compare with [6], one must consider $\epsilon = 1$, and all flows having the same maximal packet size l^m. Under these assumptions, $(L^{\epsilon} - l_i^{m\text{-}\epsilon}) = (n-1)(l^m - 1)$ and $l_i^{m\text{-}\epsilon} = l_i^m - 1$. Then θ_i defined in Eq. (6) becomes

$$\begin{aligned} r\theta_i &= (n-1)(l^m-1) + (F - Q_i)(1 + \frac{l^m - 1}{Q_i}) \\ r\theta_i^{[6]} &= \left(\frac{F}{Q_i} + n - 2\right)(l^m - 1) + (F - Q_i) \\ r\theta_i^{[6]} - r\theta_i &= (\frac{F}{Q_i} - 1)(l^m - 1) - (F - Q_i)\frac{l^m - 1}{Q_i} \\ &= 0 \end{aligned}$$

That is to say, assuming (as a restrictive hypothesis) that the server has a constant rate, all flows have the same maximal size, and packets and quanta are integer multiple of one bit, our new result is as good as [6]. If, instead, maximum packet sizes are different, the latency increases with the difference between $(n-1)(l^m - 1)$ and $\sum_{j \neq i} l_j^m - \epsilon$.

To compare with [9], the ϵ term must be removed, hence $L^{\epsilon} = L$ and $l^{m\text{-}\epsilon} = l^m$. In this case, θ_i becomes:

$$r\theta_i = (F - Q_i)\left(1 + \frac{l_i^m}{Q_i}\right) + \sum_{j=1}^{n} l_j^m - l_i^m \quad (7)$$

and it directly gives

$$r(\theta_i^{[9]} - \theta_i) = l_i^m \quad (8)$$

The above result should not be misconstrued to imply that our latency is better than the one in [9] by a factor of $\frac{l_i^m}{r}$. In fact, the gap originates from different definitions of the term "latency". In our case (following standard network calculus lexicon) it is the time before a flow gets some service. In [9], instead (following the lexicon adopted in [17] and other works of the same authors), it is meant as the time before a flow transmits a packet, which requires that the flows receives at least l_i^m units of service. Hence it also includes a factor $\frac{l_i^m}{r}$ to account for maximum-length packetisation.

V. Overall pessimism evaluation

The overall pessimism of our approach can be evaluated by measuring the distance between the original service curve β and the sum of all the residual service curves.

Let us consider a server S shared by n flows $(R_1, \dots, R_n) \xrightarrow{S} (R'_1, \dots, R'_n)$, offering a strict service β. It means that, for all backlogged period $[s, t)$:

$$\sum_{i=1}^{n} R'_i(t) - R'_i(s) \geq \beta(t - s) \quad (9)$$

Assume that, for each flow, a residual strict service curve β_i can be derived, then for all R_i backlogged period $[s_i, t_i)$:

$$R'_i(t_i) - R'_i(s_i) \geq \beta_i(t_i - s_i) \quad (10)$$

Now, consider that $[s, t)$ is a backlogged period for each R_i (for example, all flow send packets at time s, and t is the first instant when one first queue is empty), then they are two bounds:

$$\begin{aligned} \sum_{i=1}^{n} R'_i(t) - R'_i(s) &\geq \beta(t - s) \\ \sum_{i=1}^{n} R'_i(t) - R'_i(s) &\geq \sum_{i=1}^{n} \beta_i(t - s) \end{aligned}$$

Of course, $\sum_{i=1}^{n} \beta_i(t-s) \leq q\beta(t-s)$ (otherwise, decomposing a service would increase the capacity, which is impossible). The overall pessimism given by decomposition into residual service curves can be defined as:

$$\text{Pess} = \beta - \sum_{i=1}^{n} \beta_i \quad (11)$$

Then, the overall pessimism of Theorem 2 is:

$$\text{DRR-Pess} = \beta - \sum_{i=1}^{n} \beta_i^{\text{DRR}} \quad (12)$$

To simplify notation, the β_i^{DRR} is considered instead of $\beta_i^{\epsilon\text{-DRR}}$ (we have already shown that the impact of the ϵ term is negligible in practice).

$$\begin{aligned} & F(\beta - \sum_{i=1}^{n} \beta_i^{\text{DRR}}) \\ \geq & F\left(\beta - \sum_{i=1}^{n} \frac{Q_i}{F}\beta + \sum_{i=1}^{n} \frac{Q_i(L - l_i^m) + (F - Q_i)(Q_i + l_i^m)}{F}\right) \\ = & \sum_{i=1}^{n} (Q_i(L - l_i^m) + (F - Q_i)(Q_i + l_i^m)) \\ = & \sum_{i=1}^{n} (Q_i L - 2Q_i l_i^m + FQ_i + Fl_i^m - Q_i^2) \\ = & F^2 + 2FL - 2\sum_{i=1}^{n}(Q_i l_i^m) - \sum_{i=1}^{n} Q_i^2 \\ = & \left(\sum_{i=1}^{n} Q_i\right)^2 + 2\sum_{i=1}^{n} Q_i \sum_{i=1}^{n} l_i^m - 2\sum_{i=1}^{n}(Q_i l_i^m) - \sum_{i=1}^{n} Q_i^2 \\ = & \left(\sum_{i=1}^{n} Q_i + l_i^m\right)^2 - \sum_{i=1}^{n}(Q_i + l_i^m)^2 + \left(\sum_{i=1}^{n} l_i^m\right)^2 - \sum_{i=1}^{n}(l_i^m)^2 \end{aligned}$$

Hence, we get

$$\begin{aligned} & \text{DRR-Pess} \\ \geq & \frac{1}{F}\left[\left(\sum_{i=1}^{n} Q_i + l_i^m\right)^2 - \sum_{i=1}^{n}(Q_i + l_i^m)^2 + \left(\sum_{i=1}^{n} l_i^m\right)^2 - \sum_{i=1}^{n}(l_i^m)^2\right]^+ \end{aligned}$$

Now, since $Q_i \geq l_i^m$, this term has lower bound:

$$\text{DRR-Pess} \geq 3\frac{\left(\sum_{i=1}^{n} l_i^m\right)^2 - \sum_{i=1}^{n}(l_i^m)^2}{\left(\sum_{i=1}^{n} Q_i\right)} \quad (13)$$

This lower bound is reached when $Q_i = l_i^m$. If all l_i^m are the same and equal to l^m, it becomes:

$$\text{DRR-Pess} \geq nl^m - \frac{l^m}{n} \tag{14}$$

It confirms the intuition that the algorithm is somehow optimal when $Q_i = l_i^m$, and shows that our analysis somehow "loses" one frame per flow, leading to a pessimism increasing linearly with the number of flows. This was expectable, since residual service curves provide a measure of a worst-case service for each flow i. Such worst-case service takes place when a whole frame (minus one quantum) expires before flow i gets serviced, and worst-case scenarios cannot happen simultaneously for all the flows. Therefore, the overall pessimism should indeed grow by one frame per flow.

VI. Conclusions

Deficit Round Robin is a common policy used to share the capacity of a server in a fair way. It has good fairness properties and constant per-packet complexity, but a non negligible latency, that must be evaluated as accurately as possible.

This paper presents an evaluation of the DRR policy using network calculus under very general assumptions. This framework generalises previous studies, taking into account:

- per flow maximal packet size (unlike [6]),
- some ϵ term modelling the fact that packet lengths are discrete (unlike [9]), and
- a server that exhibits any kind of (strict) service curve, which generalises both the above works, which assume constant-rate servers.

A per-flow residual service curve is given. It is proved that its latency is tighter than (or as tight as) that of previous works, when evaluated under the hypotheses of these works, and is therefore better in general.

Two sequels are planned for this work: one on NP-SP/DRR integration and one on enhancing the same results.

As for the first issue, we have already stated that, due to its non negligible latency, DRR cannot be used for flows with very low delay constraints. Then, one can envisage a hierarchical scheduling framework with three priority levels: the top priority for very low-latency flows, the second for some real-time flows, sharing the available capacity using a DRR policy, and the last for best-effort, non-critical flows. In network calculus, this can be done by mixing the results on NP-SP [13], [14] and the current one on DRR. Such kind of work was partly undertaken in [12], where commercial variants of the DRR are evaluated. However, that work cannot be easily generalised, and we expect that the compositional properties of network calculus will yield a relatively simple proof under more general conditions.

Each time a network calculus bound is given, the first question that arises is "is the residual service tight?" *i.e.* "is it the exact worst case, or only a bound?". Our intuition, following the work done in [13], [14], is that the current bound can be enhanced. If the shared service offers a service curve $\beta = \lambda_r$, then, Theorem 2 yields a residual service in the form of a rate-latency curve of rate $\rho_i r$ (with $\rho_i = \frac{Q_i}{\sum_j Q_j}$). However, in practice, the real output is a shaped stair case, as illustrated in Figure 5, as a flow alternates between being served (*i.e.*, being given the full server capacity, whatever the server's service curve) and waiting for its round-robin turn. In Figure 5, the $\rho_i\beta - X_i$ is the current result, and the alternate red curve is the one that is expectable in practice.

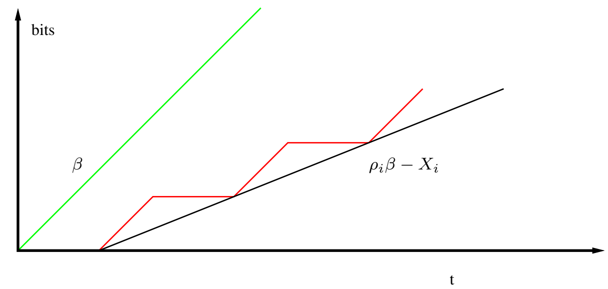


Fig. 5. Enhancement intuition

VII. Acknowledgement

Thanks to Guillaume Dufour for help on Section V.

References

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

[2] A. Bouillard, L. Jouhet, and E. Thierry. Comparison of different classes of service curves in network calculus. In *Proc. of the 10th International Workshop on Discrete Event Systems (WODES 2010)*, Technische Universität Berlin, August 30 - September 1 2010.

[3] M. Boyer, N. Navet, and M. Fumey. Experimental assessment of timing verification techniques for afdx. In *Proc. of the 6th Int. Congress on Embedded Real Time Software and Systems*, Toulouse, France, February 2012.

[4] J. Grieu. *Analyse et évaluation de techniques de commutation Ethernet pour l'interconnexion des systèmes avioniques*. Thèse de Doctorat, Institut National Polytechnique de Toulouse, Toulouse, 2004.

[5] 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 (Salzburg, Austria, September 30 - October 03, 2007)*, pages 173–178, New York, NY, USA, 2007. ACM.

[6] S. Kanhere and H. Sethu. On the latency bound of deficit round robin. In *Proc. of 11th Int. Conf. on Computer Communications and Networks (ICCCN 2002)*, pages 548 – 553, oct. 2002.

[7] J.-Y. Le Boudec and P. Thiran. *Network Calculus*, volume 2050 of *LNCS*. Springer Verlag, 2001. http://lrcwww.epfl.ch/PS_files/NetCal.htm.

[8] L. Lenzini, E. Mingozzi, and G. Stea. Aliquem: a novel DRR implementation to achieve better latency and fairness at O(1) complexity. In *Proc. of 10th IEEE Int. Workshop on Quality of Service (IWQoS 2002)*, pages 77 – 86, 2002.

[9] L. Lenzini, E. Mingozzi, and G. Stea. Full exploitation of the deficit round robin capabilities by efficient implementation and parameter tuning. Technical report, Dipartimento di Ingegneria della Informazione, University of Pisa, October 2003.

[10] L. Lenzini, E. Mingozzi, and G. Stea. Delay bounds for FIFO aggegates: a case study. *Computer Communications*, 28:287–299, 2004.

[11] L. Lenzini, E. Mingozzi, and G. Stea. Tradeoffs between low complexity, low latency and fairness with deficit round robin schedulers. *IEEE/ACM Transactions on Networking*, 12(4):681–693, August 2004.

[12] L. Lenzini, E. Mingozzi, and G. Stea. Performance analysis of modified deficit round robin schedulers. *IOS Journal of High Speed Networks*, 16(4):399–422, 2007.

[13] W. Mangoua Sofack and M. Boyer. Non preemptive static priority with network calculus. In *Proc. of the 16th IEEE int. conf. on Emerging Technologies and Factory Automation (ETFA'11)*, September 2011.

[14] W. Mangoua Sofack and M. Boyer. Non preemptive static priority with network calculus: Enhanced. In *Proc. of the 16th International GI/ITG Conference on Measurement, Modelling and Evaluation of Computing Systems and Dependability and Fault Tolerance (MMB & DFT 2012) – Workshop on Network Calculus (WoNeCa)*, March 2012.

[15] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking (TON)*, 1, 1993.

[16] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. *SIGCOMM Computer Communication*, 25:231–242, October 1995.

[17] D. Stiliadis. *Traffic Scheduling in Packet-Switched Networks: Analysis, Design and Implementation*. PhD thesis, University of California, Santa-Cruz, June 1998.

[18] D. Stiliadis and A. Varma. Latency-rate servers: a general model for analysis of traffic scheduling algorithms. *IEEE/ACM Transactions on Networking*, 6(5):611–624, oct 1998.

[19] H. Yu and X. Liu. Scheduling design and analysis for end-to-end heterogeneous flows in an avionics network. In *Proc. of the 30st IEEE Int. Conf. on Computer Communications (IEEE INFOCOM 2012)*, pages 2417 –2425, april 2011.

APPENDIX

We give here the full proof of the above theorem, together with intermediate results and definitions. In order to simplify the notation, if R is a cumulative function for a flow, notation $R(s,t)$ is used for $R(t) - R(s)$.

The rest of the proof assumes that $\beta \neq 0$ (otherwise, $\beta_i^{\text{DRR}} = \beta_i^{\epsilon\text{-}DRR}$ and the Theorem 2 is trivial).

Lemma 1 (Service curve grows to infinity):

$$\beta \neq 0, \beta \text{ curve of strict service } \implies \lim_{x\to\infty} \beta(x) = \infty \quad (15)$$

Proof: If β is the curve of a strict service, it is super-additive [2]. $\beta \neq 0 \implies \exists x, \beta(x) > 0$ and, for all n, $\beta(nx) > nx$. ∎

This Lemma is necessary to prove that, for each instant s, it the i-th flow is backlogged, it will be served in the future (at each iteration of the loop, only a finite quantity of other flows is served, and then, the i-th flows will be served).

Definition 1: DRR trace definition Let $[u, v[$ be a maximal backlog period. Let $(\tau_k, fl_k)_{k\in[0,N]}$ be the sequence of couples (instant, flow), printed at line 6 in the algorithm. ($u = \tau_0$). The sequence is completed by $\tau_{N+1} = v$.

Note that the τ_k sequence is increasing ($\tau_k < \tau_{k+1}$), since the `send` instruction has a non-null duration (Section III-B).

From this sequence, three kinds of objects are derived: function $fl : [u, v[\mapsto [1, n]$ that gives the flow currently served at each instant, and, for each flow i, sequences τ_j^i and η_j^i, describing the start and end of j-th service opportunity. These quantities are shown in Figure 6.

Function $fl : \mathbb{R} \to [1, n]$ is a time projection of the fl_k sequence, describing which flow is served at each time instant.

$$fl(t) \stackrel{\text{def}}{=} fl_k \text{ with } k = \max\{k' \mid \tau_{k'} \leq t\}$$

For each flow i, τ_j^i is the start of the j-th service opportunity, *i.e.* τ_k such that $fl_k = i$. Formally, if a backlog instant t exists in $[u, v[$ for flow i ($R'(t) > R(t)$), then

$$\tau_0^i \stackrel{\text{def}}{=} \min\{\tau_k \mid fl_k = i\}$$
$$\tau_j^i \stackrel{\text{def}}{=} \tau_{k_j^i} \text{ with } k_j^i \text{ s.t. } j = \left|\left\{\tau_p \,\middle|\, fl_p = i, p \leq k_j^i\right\}\right|$$

For a given i, k_j^i is an increasing sub-sequence of $\mathbb{N}$. The η_j^i instants mark the ends of service opportunities:

$$\eta_j^i \stackrel{\text{def}}{=} \tau_{k+1} \text{ with } \tau_j^i = \tau_k$$

The τ_i^j instants defined here are the same instants as the τ_k^i defined in [6], with i being an index on the flow, and j, k being sequence indexes.

The model assumes that, on $[\tau_i, \tau_{i+1}[$, flow fl_i is served, an no one else.

$$\forall \tau_k, \forall j \neq fl_k, \forall t \in [\tau_k, \tau_{k+1}[: R'_j(\tau_k, t) = 0 \qquad (16)$$

Lemma 2 (Bounds on Deficit counter): Let $[u, v[$ be a maximal backlog period, and τ_k, fl_k, τ_k^i as defined in Def. 1. Let $DC(i, \tau_k)$ be the value of $\mathtt{DC}[i]$ when the code passes line 6 at time τ_k. Then, the following properties hold [16].

$$\forall \tau_j^i : 0 \leq DC(i, \tau_j^i) < l_i^m \qquad (17)$$
$$\forall \tau_j^i : 0 \leq DC(i, \tau_j^i) \leq l_i^m - \epsilon \qquad (18)$$
$$\forall \tau_j^i : DC(i, \tau_{j+1}^i) - DC(i, \tau_j^i) \leq Q_i \qquad (19)$$
$$fl(\eta_j^i) \neq i \implies R'_i(\tau_j^i, \eta_j^i) \leq Q_i + DC(i, \tau_j^i) - DC(i, \tau_{j+1}^i) \qquad (20)$$

which leads to

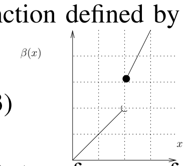
$$\forall t \in [\eta_{j+p-1}^i, \tau_{j+p}^i) : R'_i(\tau_j^i, t) \leq pQ_i + l_i^m - \epsilon < pQ_i + l_i^m \qquad (21)$$

One can also focus on backlog period

$$[\tau_k^i, \tau_{k+p}^i[\ i\text{-backlog period} \Rightarrow R'_i(\tau_k^i, \tau_{k+p}^i) \geq pQ_i - l_i^m - \epsilon > pQ_i - l_i^m \qquad (22)$$

a) Remark on equations 20 and 21: The equations 20 and 21 have the precondition $fl(\eta_j^i) \neq i$, meaning that the property only holds if there is not two consecutive services of the same flow (meaning that the i flow is the only active one). This condition is related to continuity of β at scheduling point. Let consider $l_1^m = Q_1$, and a β function defined by

$$\beta(x) = \begin{cases} x & \text{if } x < Q_1 \\ 2x - \frac{Q_1}{2} & \text{if } x \geq Q_1 \end{cases} \qquad (23)$$



Then, consider the case when, at time 0, two frames of the size l_1^m of the first flow are put in the first queue. Assume that the server output is exactly β. At first iteration, $\mathtt{DC}[1] = 0$, and $DC(\tau_0^1) = 0$. The first frame is send from time 0 to Q_i ($\tau_0^1 = 0, \eta_0^1 = Q_1$), and the deficit counter is decremented by the size of the send frame *i.e.* Q_1. Then, a second iteration begins, with $DC(1, \tau_1^1) = 0$. But, at this instant $\tau_1^1 = Q_i$ the server begins to serve the second frame, and, outputs half of this frame instantaneously. Then, $R'(\tau_0^1, \eta_0^1) = \frac{3}{2}Q_i > Q_i +$

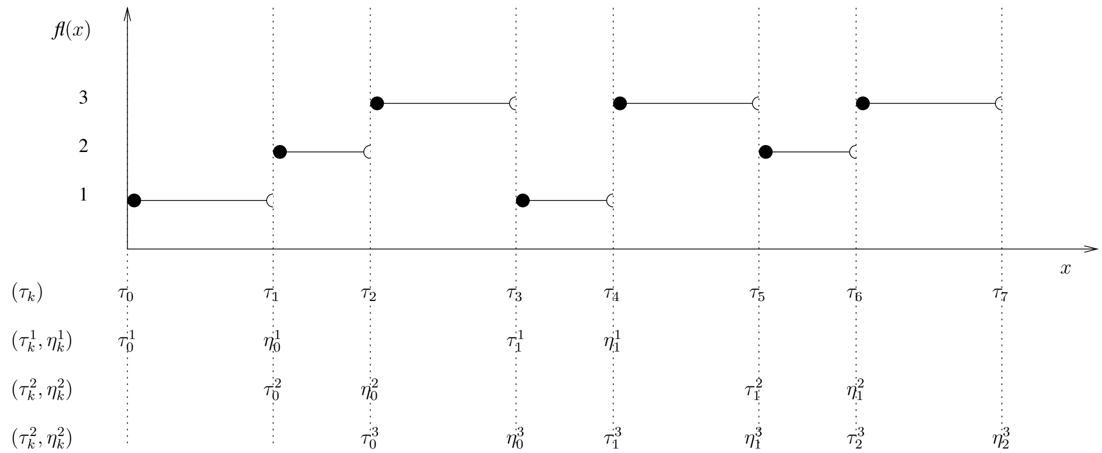


Fig. 6. Illustration of $fl(x)$, τ_j^i, η_j^i

$DC(\tau_0^1) - DC(\tau_1^1)$. This could have been solved introducing a notion of successive actions at the same instant, but the condition $fl(\eta_j^i) \neq i$, meaning that the property holds if the same flow is not servers two times in sequence is sufficient for the proof.

of Lemma 2: Relations 17 and 20 were already in [16, Lemma 4.1 and Theorem 4.2]. Proofs are also given here for ease of reading.

Proof of Eq. 17–18. The $\texttt{DC}[i]$ variable is initialised to 0, and decremented in one line only in the algorithm. Before decrementing the variable $\texttt{DC}[i]$ by size(head(i)), it is checked that size(head(i)) $\leq \texttt{DC}[i]$, which ensures $\texttt{DC}[i] \geq 0$.

Conversely, $\texttt{DC}[i]$ is incremented only if the queue is not empty, and, in this case, while $\texttt{DC}[i] \geq size(head(i))$, it is decremented. When the loop ends, $\texttt{DC}[i] < size(head(i)) \leq l_i^m$, which yields Eq. 17.

Proving bound $\texttt{DC}[i] \leq l_i^m - \epsilon$ needs an intermediate result: $\texttt{DC}[i]$ is always a multiple of ϵ.

The above statement is true at reset (lines 1 and 14), and preserved when incrementing the variable by Q_i ($\texttt{DC}[i] \leftarrow \texttt{DC}[i] + Q_i$) – since Q_i is also a multiple of ϵ, and when decrementing the variable ($\texttt{DC}[i] \leftarrow \texttt{DC}[i]$ – size(head(i))) – since packet sizes are multiple of ϵ. Then, from $\texttt{DC}[i] < l_i^m$, and $\texttt{DC}[i]$ and l_i^m are multiple of ϵ, it follows that $\texttt{DC}[i] \leq l_i^m - \epsilon$.

Proof of Eq. 19. $\texttt{DC}[i]$ is incremented only once per iteration, by quantum Q_i.

Proof of Eq. 20. Let us introduce $DC'(i, \tau_k^i)$ as the value of $\texttt{DC}[i]$ between lines 13 and 14 ($DC'(i, \tau_k^i) \geq 0$). Since the $\texttt{DC}[i]$ variable is decremented by the amount of sent data, it follows that:

$$R_i'(\tau_j^i, \eta_j^i) = Q_i + DC(i, \tau_j^i) - DC'(i, \tau_j^i) \qquad (24)$$

And $DC(i, \tau_{j+1}^i) = DC'(i, \tau_j^i)$ or $DC(i, \tau_{j+1}^i) = 0$ depending on test at line 14. Then, $DC(i, \tau_{j+1}^i) \leq DC'(i, \tau_j^i)$.

Proof of Eq. 21. It is a simple consequence of Eq. 20, giving the bound on one service period, Eq. 16, ensuring that there is not output between two period of service, and 18 giving bounds on $DC(\cdot, \cdot)$ values.

$$\begin{aligned} R_i'(\tau_j^i, t) &= \sum_{k=0}^{p-1} R_i'(\tau_{j+k}^i, \tau_{j+k+1}^i) \\ &\leq \sum_{k=0}^{p-1} (Q_i + DC(i, \tau_{j+k}^i) - DC(i, \tau_{j+k-1}^i)) \\ &= (\sum_{k=0}^{p-1} Q_i) + DC(i, \tau_j^i) - DC(i, \tau_{j+k-1}^i) \\ &\leq pQ_i + l_i^m - \epsilon \end{aligned}$$

Proof of Eq. 22.

The same decomposition as in the previous proof is done, however using eq. 24 instead of 20. We also use the fact that, in a backlogged period of flow i, $DC'(i, \tau_j^i) = DC(i, \tau_{j+1}^i)$. Then, sum $\sum_{k=j}^{j+p-1}$ of eq. 24 leads to $R_i'(\tau_j^i, \tau_{j+p}^i) = pQ_i + DC(i, \tau_j^i) - DC'(i, \tau_{j+p-1}^i)$, and using eq. 18 both DC and DC' can be removed. ■

Lemma 3 (Number of cycles): Let s be a instant in a backlogged period of i, within the maximal general backlog period $[u, v[$. Let τ_k^i be the start of the next period of service ($\tau_k^i = \min \left\{ \tau_m^i \mid \tau_m^i \geq s \right\}$). Then, each other flow has been served at most $p + 1$ times between s and τ_{k+p}^i.

$$\forall j \neq i, \left| \left\{ \tau_m^j \mid s \leq \tau_m^j < \tau_{k+p}^i \right\} \right| \leq p + 1 \qquad (25)$$

Proof: Since the loop is always executed in the same order, and since $[s, \tau_{k+p}^i[$ is a backlog period for R_i, the condition at line 6 is always true. Then, each flow is selected at most once in $[s, \tau_k^i[$, they are p iterations of the loop between τ_k^i and τ_{k+p}^i and each other flow is selected only once per iteration. ■

of Theorem 2: Let i be a flow, with input function R_i and output function R_i'. Let $s \leq t$ be two instants in its backlogged period. Let $[u, v[$ be the maximal backlog period of the system which includes s and t. Let us consider the sequences τ_i, fl_i τ_i^j, as defined in Def.1.

Let τ_k^i be the start of the first service opportunity for flow i after s (there is one for sure, since s marks a backlogged period of i and DRR is work conserving), $\mathcal{P}$ the set of *complete* service opportunities of i between s and t, and p its cardinality (if there is no service opportunity between s and t, $\mathcal{P} = \emptyset$ and $p = 0$).

$$\begin{aligned}
\tau_k^i &\overset{\text{def}}{=} \min\left\{\tau_m^i \mid \tau_m^i \geq s\right\} \\
\mathcal{P} &\overset{\text{def}}{=} \{\tau_m \mid s \leq \tau_m, \tau_{m+1} \leq t, \mathit{fl}_m = i\} \\
&= \left\{\tau_j^i \,\middle|\, s \leq \tau_j^i, \eta_j^i \leq t\right\} \\
p &= |\mathcal{P}|
\end{aligned}$$

Sub-goal 1. $R_i'(s,t) \geq pQ_i - l^m - \epsilon$

Assume $p > 0$ (the case $p = 0$ is postponed). In this case, $\mathcal{P} = \left\{\tau_k^i, \ldots, \tau_{k+p-1}^i\right\}$. Since $\tau_k^i \geq s$, and $R_i' \in \mathcal{F}$ is non decreasing, we have

$$R_i'(\tau_k^i) \geq R_i'(s) \tag{26}$$

Let us consider τ_{k+p}^i.

If t is within a service opportunity of i ($\mathit{fl}(t) = i$), then $\tau_{k+p}^i \leq t$ ($\mathit{fl}(t) = i$ implies that there exists k' such that $\tau_{k'}^i \leq t < \eta_{k'}^i$, then $\tau_{k'}^i \notin \mathcal{P}$ and then $k' > k+p-1$, *i.e.* $k \geq k+p$ and since τ^i is increasing, $t \geq \tau_{k+p}^i$) $R_i'(\tau_{k+p}^i) \leq R_i'(t)$.

If t is outside a service opportunity of i, $\tau_{k+p}^i > t$ then $R_i'(\tau_{k+p}) = R_i'(t)$ (by successive applications of eq.16) Then, in both cases:

$$R_i'(\tau_{k+p}^i) \leq R_i'(t) \tag{27}$$

Then, $R_i'(s,t) = R_i'(t) - R_i'(s) \geq R_i'(\tau_{k+p}^i) - R_i'(\tau_k^i) \geq pQ_i - l_i^m - \epsilon$ from Eq. 22 (the backlog hypothesis $[\tau_k^i, \tau_{k+p}^i[$ comes from the fact that $[\tau_k^i, \tau_{k+p}^i[\subset [s,t[$ and $[s,t[$ is a backlogged period).

If $p = 0$, $pQ_i - l_i^m - \epsilon \leq 0 \leq R_i'(s,t)$.

This proves Sub-goal 1, *i.e.* a lower bound on $R_i'(s,t)$ as a function of p:

$$R_i'(s,t) \geq pQ_i - l_i^m - \epsilon \tag{28}$$

Now, one has to find a lower bound on p based on $\beta(t-s)$.

Sub-goal 2. $\frac{\beta(t-s) - (R_i'(s,t) + \sum_{j\neq i}(l_j^m - \epsilon))}{\sum_{j\neq i} Q_j} \leq p+1$

Since the server offers a strict service curve, the following holds:

$$\beta(t-s) \leq R'(s,t) = R_i'(t) - R_i'(s) + \sum_{j\neq i} R_j'(t) - R_j'(s)$$

If t is within a service opportunity of i, $\tau_{k+p}^i \leq t$, however, for all $j \neq i$, $R_j'(\tau_{k+p}^i) = R_j'(t)$ (by successive applications of eq. 16). Otherwise, $\tau_{k+p}^i \geq t$ and $R_j'(\tau_{k+p}^i) \geq R_j'(t)$. Then, in both cases $R_j'(\tau_{k+p}^i) \geq R_j'(t)$.

$$\beta(t-s) \leq R_i'(t) - R_i'(s) + \sum_{j\neq i} R_j'(\tau_{k+p}^i) - R_j'(s)$$

By Lemma 3, there are at most $p+1$ cycles for each R_j between s and $R_j'(\tau_{k-1}^i)$, *i.e.* $R_j'(\tau_{k+p}^i) - R_j'(s) \leq (p+1)Q_j + l_j^m - \epsilon$ (using eq. 21).

A simple sum yields a lower bound on p expressed as a function of β, Q_j, l_j^m and $R_i'(s,t)$.

$$\beta(t-s) \leq R_i'(s,t) + \sum_{j\neq i}\left((p+1)Q_j + l_j^m - \epsilon\right) \tag{29}$$

$$= R_i'(s,t) + (p+1)\sum_{j\neq i} Q_j + \sum_{j\neq i}\left(l_j^m - \epsilon\right) \tag{30}$$

Now, all it takes is some straightforward algebraic manipulation of eq. 28 and eq. 29.

To simplify the notation, we use $F = \sum_{j=1}^n Q_j$, $l_i^{m\text{-}\epsilon} = l_i^m - \epsilon$, and $L^\epsilon = \sum_{j=1}^n l_i^{m\text{-}\epsilon}$. Then, $\sum_{j\neq i} Q_i = F - Q_i$, and $\sum_{j\neq i} l_j^m - \epsilon = L^\epsilon - l_i^{m\text{-}\epsilon}$.

$$\begin{aligned}
& \textit{eq. } 29 \\
\iff & \beta(t-s) - (R_i'(s,t) + (L^\epsilon - l_i^{m\text{-}\epsilon})) \leq (p+1)(F - Q_i) \\
\iff & \frac{\beta(t-s) - R_i'(s,t) - (L^\epsilon - l_i^{m\text{-}\epsilon}) - (F - Q_i)}{F - Q_i} \leq p
\end{aligned}$$

Reporting this bound on p in eq. 28 yields:

$$\begin{aligned}
R_i'(s,t) \geq\ & \frac{\beta(t-s) - R_i'(s,t) - (L^\epsilon - l_i^{m\text{-}\epsilon}) - (F - Q_i)}{F - Q_i} Q_i - l_i^m \\
\iff & (F - Q_i)R_i'(s,t) \geq Q_i\beta(t-s) - Q_iR_i'(s,t) \\
& - Q_i\left((L^\epsilon - l_i^{m\text{-}\epsilon}) + (F - Q_i)\right) - (F - Q_i)l_i^{m\text{-}\epsilon} \\
\iff & FR_i'(s,t) \geq Q_i\beta(t-s) \\
& - Q_i\left((L^\epsilon - l_i^{m\text{-}\epsilon}) + (F - Q_i)\right) - (F - Q_i)l_i^{m\text{-}\epsilon} \\
\iff & R_i'(s,t) \geq \frac{Q_i}{F}\beta(t-s) - \frac{Q_i(L^\epsilon - l_i^{m\text{-}\epsilon}) + (F - Q_i)(Q_i + l_i^{m\text{-}\epsilon})}{F}
\end{aligned}$$

Moreover, since $R_i'(t)$ is non decreasing, $R_i'(t) - R_i'(s) \geq 0$, then, the operator $[\cdot]^+$ can be applied. ∎