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—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 is such a theory, with a strong mathematical background, and some success stories: it has been used to certify the A380 AFDX backbone. 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% Moreover, people often restrict network calculus to (σ, ρ)-calculus, i.e., a calculus were server only offer rate-latency capacity (βr,x service curves in network calculus), and flows are constrained by token buckets (γr,b arrival curves – see Section I for details). However the theory allows other types of curves, offering a better modelling of flows and then more realistic bounds.

In particular, a recent work 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, 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 V 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 (+, ×) 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 $[\ ]^+: x \mapsto \max(x, 0)$.

1This work has been partially funded by French ANR agency under project id ANR-09-SEGI-009.
• **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 $f^+(t) = \sup_{0 \leq s \leq t} f(s)$

To model flows constraint and service guarantees, network calculus uses a set of usual parametrised curves, $\delta_d$, $\lambda_R$, $\beta_R$, $\gamma_{r,b}$, $\nu_{T,T}$, defined by:

\[
\delta_d(t) = \begin{cases} 
0 & \text{if } t \leq d \\
\infty & \text{otherwise}
\end{cases}
\]
\[
\gamma_{r,b}(t) = \begin{cases} 
rt + b & \text{if } t > 0 \\
0 & \text{otherwise}
\end{cases}
\]
\[
\lambda_R(t) = Rt \\
\beta_R(t) = R[t - T]^+
\]
\[
\nu_{T,T}(t) = \min \left( \delta_0, \left\lceil \frac{t + \tau}{T} \right\rceil \right)
\]

**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 $R \overset{S}{\rightarrow} R'$, 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.

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:

\[
StBl(t) = \sup \{u \leq t \mid R'(u) = R(u)\}
\]

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

The virtual delay 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:

\[
d(t) = \inf \{\tau \geq 0 \mid R(t) \leq R'(t + \tau)\}
\]

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.

---

2Some 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.
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 $\alpha$ as an arrival curve. This condition is equivalent to $R \leq R \ast \alpha$.

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

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 $\beta$ if and only if, for all pair $R \xrightarrow{S} R$, $R' \geq R \ast \beta$.

We say that a system $S$ offers a strict service curve $\beta$ 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 1.3.1, 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 $\alpha$, 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: $d(t) \leq h(\alpha,\beta)$.

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: $S \in \mathcal{F}^n \times \mathcal{F}^n$, $(R_1,\ldots,R_n) \xrightarrow{S} (R'_1,\ldots,R'_m)$. And the capacity of the server is shared by several flows.

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

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 $\beta$ 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. $R_1' + R_2' \geq (R_1 + R_2)' \ast \beta$), but the repartition of the service among the flows depends on the server policy (common policies are FIFO [10], static priorities [4, 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 $\beta_i$ 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 curve\footnote{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.}. 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 $\beta$ (with $\sum \rho_i = 1$). 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 fractional 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 ($\beta = \lambda_n$), but the definition can be generalised to any kind of $\beta$ service curve in the context of network calculus: the generalisation of GPS just states that the aggregated service $\beta$ 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
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 not empty, it is selected, the Deficit Counter $DC[i]$ is increased by $Q_i$, called the flow’s quantum, and as long as $DC[i]$ is greater than or equal to the size of the head-of-line packet, the packet is sent and the $DC[i]$ 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, $DC[i]$ 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 $\text{now}()$, 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.

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 $\beta$, shared by $n$ flows $R_1, \ldots, R_n$. 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 $\beta_i^{\text{DRR}}$, defined by:

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

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

Input: Per flow quantum: $Q_1, \ldots, Q_n$ (Integer)
Data: Per flow deficit: $DC[1..n]$ (Integer)
Data: Counter: $k$ (Integer)

```
for $i = 1$ to $n$ do
    $DC[i] \leftarrow 0$
end

while True do
    for $i = 1$ to $n$ do
        if not empty($i$) then
            // Print is a pseudo-instruction, used to simplify the proof
            print (now(), $i$);
            $DC[i] \leftarrow DC[i] + Q_i$
            while (not empty($i$)) and ($size(head(i)) \leq DC[i]$) do
                send(head($i$));
                $DC[i] \leftarrow DC[i] - size(head(i))$
                removeHead($i$)
            end
            if empty($i$) then
                $DC[i] \leftarrow 0$
            end
        end
    end
end
```

Algorithm 1: DRR algorithm

Moreover, if the packet length is discrete, multiple of a basic unit $\epsilon$ (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:

$$\beta_i^{\epsilon-\text{DRR}}(t) = \left[ \frac{Q_i}{F} \beta(t) - \frac{Q_i(L' - l_i^m - \epsilon) + (F - Q_i)(Q_i + l_i^m - \epsilon)}{F} \right]^+$$

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

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 $DC[i]$ always lies within an interval $[0, l_i^m + Q_i]$ (and, in fact, within $[0, l_i^m]$ when considering values at line [9], 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 ser-
vices of $i$ between $s$ and $t$. Then, $\sum_{j \neq i} R'_i < \sum_{j \neq i} ((p+1)Q_j + l'_m) = (p+1) \sum_{j \neq i} Q_j + \sum_{j \neq i} l'_m$, which yields a lower bound for $p$. It is also known that $R'_i(s, t) \geq pQ_i - l'_m$. 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]. The basic unit $\epsilon$ has been introduced to ensure comparison with [6]. If the packet length and quanta are discrete, the interval within $\text{DC}[i]$ always falls, which were reported to be $[0, l'_m + Q_i]$ and $[0, l'_m]$, can instead be rewritten as $[0, l'_m - \epsilon + Q_i]$ and $[0, l'_m - \epsilon]$. This allows one to save an $\epsilon$ worth of latency for each flow wherever an $l'_m$ term appears. The expected gain of modelling quanta and packets as discrete entities (rather than continuous) is however negligible in practical cases, where $\epsilon$ is one byte and $l'_m$ 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 [13], with a low implementation complexity ($O(1)$ if each quantum $Q_i$ is no less than the maximal packet size [16].) 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 ($\lambda_i$ in network calculus), and see the residual service for a flow as a Latency-rate (or rate-latency) server [18]. ($\beta_{H,T}$ in network calculus). In this framework, all papers show that the residual service for flow $i$ is a $\beta_{\frac{Q_i}{r}, \theta_i}$, and each paper provides its own evaluation of $\theta_i$.

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

$$\theta_i^{[17]} = \frac{3F - 2Q_i}{r} \quad (3)$$

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:

$$\theta_i^{[6]} = \frac{1}{r} \left( F - Q_i + (l'_m - 1) \left( \frac{F}{Q_i} + n - 2 \right) \right) \quad (4)$$

with $l'_m = \max_j \{ l'_m \}$. Independently, a third bound was found in [8], [9]:

$$\theta_i^{[8]} = \frac{1}{r} \left( (F - Q_i)(1 + \frac{l'_m}{Q_i}) + \sum_{j=1}^n \frac{l'_m}{Q_j} \right) \quad (5)$$

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 $\beta(t) = \lambda_i t = rt$, even though the result of Theorem 2 is more general. Considering that $\beta = \lambda_r$, our result gives:

$$\theta_i = \frac{1}{r} \left( L' - l'_m + (F - Q_i) \left( 1 + \frac{l'_m}{Q_i} \right) \right) \quad (6)$$

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

$$\beta_i(t) = \left[ \frac{Q_i}{F} \left( t - \frac{Q_i}{F} (L' - l'_m) + (F - Q_i) \frac{Q_i + l'_m}{Q_i} \right) \right] +$$

$$= \frac{Q_i}{F} \left[ t - \frac{1}{r} \left( L' - l'_m + (F - Q_i) \frac{Q_i + l'_m}{Q_i} \right) \right] +$$

$$= \beta Q_i \frac{t}{r, \theta_i} .$$

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, \( L^r = l^m - 1 \) and \( l^m - \epsilon = l^m - 1 \).

Then \( \theta_i \) defined in Eq. (6) becomes

\[
\rho_i = (n-1)(l^m-1) + (F - Q_i)(1 + \frac{l^m-1}{Q_i})
\]

\[
\rho_i^\beta = \left( \frac{F}{Q_i} + n - 2 \right) (l^m - 1) + (F - Q_i)
\]

\[
r \rho_i^\beta - \rho_i = \left( \frac{F}{Q_i} - 1 \right) (l^m - 1) - (F - Q_i) \frac{l^m - 1}{Q_i}
\]

\[
= 0
\]

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 \( \sum_{j \neq i} l^m - \epsilon \).

To compare with [9], the \( \epsilon \) term must be removed, hence \( L^r = L \) and \( l^m - \epsilon = l^m \). In this case, \( \rho_i \) becomes:

\[
r \rho_i = (F - Q_i) \left( 1 + \frac{l^m}{Q_i} \right) + \sum_{j=1}^{n} l^m_j - l^m_i
\]

and it directly gives

\[
r (\rho_i^\beta - \rho_i) = l^m_i
\]

The above result should not be misconstrued to imply that our latency is better than the one in [9] by a factor of \( \frac{l^m_i}{l^m} \). 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 \( l^m_i \) units of service. Hence it also includes a factor \( \frac{l^m_i}{r} \) 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 \( \beta \) and the sum of all the residual service curves.

Let us consider a server \( S \) shared by \( n \) flows \( (R_1, \ldots, R_n) \) \( \leq \Rightarrow \) \( (R_1', \ldots, R_n') \), offering a strict service \( \beta \). It means that, for all backlogged period \([s, t] \):

\[
\sum_{i=1}^{n} R_i'(t) - R_i'(s) \geq \beta(t-s)
\]

(9)

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

\[
R_i'(t_i) - R_i'(s_i) \geq \beta_i(t_i-s_i)
\]

(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:

\[
\sum_{i=1}^{n} R_i'(t) - R_i'(s) \geq \beta(t-s)
\]

\[
\sum_{i=1}^{n} R_i'(t) - R_i'(s) \geq \sum_{i=1}^{n} \beta_i(t-s)
\]

(11)

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:

\[
Pess = \beta - \sum_{i=1}^{n} \beta_i
\]

Then, the overall pessimism of Theorem [2] is:

\[
\text{DRR-Pess} = \beta - \sum_{i=1}^{n} \beta_i^\text{DRR}
\]

(12)

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

\[
F(\beta - \sum_{i=1}^{n} \beta_i^\text{DRR})
\]

\[
\geq F \left( \beta - \sum_{i=1}^{n} Q_i F \beta + \sum_{i=1}^{n} Q_i (L - l^m_i) + (F - Q_i)(Q_i + l^m_i) \right)
\]

\[
= \sum_{i=1}^{n} (Q_i (L - l^m_i) + (F - Q_i)(Q_i + l^m_i))
\]

\[
= \sum_{i=1}^{n} (Q_i L - 2Q_i l^m_i + FQ_i + Fl^m_i - Q_i^2)
\]

\[
= F^2 + 2FL - 2 \sum_{i=1}^{n} (Q_i l^m_i) - \sum_{i=1}^{n} Q_i^2
\]

\[
= \left( \sum_{i=1}^{n} Q_i \right)^2 + 2 \sum_{i=1}^{n} Q_i \sum_{i=1}^{n} l^m_i - 2 \sum_{i=1}^{n} Q_i l^m_i - \sum_{i=1}^{n} Q_i^2
\]

\[
= \left( \sum_{i=1}^{n} Q_i + l^m_i \right)^2 - \sum_{i=1}^{n} (Q_i + l^m_i)^2 + \left( \sum_{i=1}^{n} l^m_i \right)^2 - \sum_{i=1}^{n} (l^m_i)^2
\]

Hence, we get

\[
\text{DRR-Pess} \geq \frac{1}{F} \left[ \left( \sum_{i=1}^{n} Q_i + l^m_i \right)^2 - \sum_{i=1}^{n} (Q_i + l^m_i)^2 + \left( \sum_{i=1}^{n} l^m_i \right)^2 - \sum_{i=1}^{n} (l^m_i)^2 \right]^{+}
\]

Now, since \( Q_i \geq l^m_i \), this term has lower bound:

\[
\text{DRR-Pess} \geq \frac{3}{F} \left[ \left( \sum_{i=1}^{n} l^m_i \right)^2 - \sum_{i=1}^{n} (l^m_i)^2 \frac{Q_i}{l^m_i} \right]
\]

(13)
This lower bound is reached when $Q_i = l^m_i$. If all $l^m_i$ are the same and equal to $l^m$, it becomes:

\[
\text{DRR-Pess} \geq n l^m - \frac{l^m}{n} \tag{14}
\]

It confirms the intuition that the algorithm is somehow optimal when $Q_i = l^m_i$, 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 $\epsilon$ 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 $\rho r$ (with $\rho = \sum_j Q_j / \sum_j l_j$). 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 $\rho_{1}\beta - X_1$ is the current result, and the alternate red curve is the one that is expectable in practice.

VII. ACKNOWLEDGEMENT

Thanks to Guillaume Dufour for help on Section V.

REFERENCES

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, \( \beta_i^{+} = \beta_i^{-} \)DRR and the Theorem 2 is trivial).

Lemma 1 (Service curve grows to infinity):

\[
\beta \neq 0, \text{ \beta curve of strict service } \Rightarrow \lim_{x \to \infty} \beta(x) = \infty
\]

Proof: If \( \beta \) is the curve of a strict service, it is superadditive [2], \( \beta \neq 0 \) \( \Rightarrow \exists x, \beta(x) > 0 \) and, for all \( n, \beta(nx) > nx \).

This Lemma is necessary to prove that, for each instant \( s \), 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 \( \tau_k \) 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 \( fl: [u,v] \rightarrow [1,n] \) that gives the flow currently served at each instant, and, for each flow \( i \), sequences \( \tau_i \) and \( \eta_i \), describing the start and end of \( j \)-th service opportunity. These quantities are shown in Figure 6.

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

\[
fl(t) \overset{\text{def}}{=} \max \{ k' \mid \tau_{k'} \leq t \}
\]

For each flow \( i \), \( \tau_i^j \) is the start of the \( j \)-th service opportunity, \( i.e. \) \( \tau_i \) such that \( fl_k = i \). Formally, if a backlog instant \( t \) exists in \([u,v]\) for flow \( i \) (\( R^i(t) > R(t) \)), then

\[
\tau_0^i \overset{\text{def}}{=} \min \{ \tau_k \mid fl_k = i \} \\
\tau_j^i \overset{\text{def}}{=} \tau_{k_j}^i \text{ with } k_j^i \text{ s.t. } j = \{ \tau_p \mid fl_p = i, p \leq k_j^i \}
\]

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

\[
\eta_j^i \overset{\text{def}}{=} \tau_{k+1}^i \text{ with } \eta_j^i = \tau_k
\]

The \( \tau_j^i \) instants defined here are the same instants as the \( \tau_j^i \) 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.

\[
\forall \tau_k, \forall j \neq fl_k, \forall t \in [\tau_k, \tau_{k+1}] : R^j_i(\tau_k, t) = 0
\]

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

\[
\forall \tau_j^i : 0 \leq DC(i, \tau_j^i) < l_i^m
\]
\[
\forall \tau_j^i : 0 \leq DC(i, \tau_j^i) \leq l_i^1 - \epsilon
\]
\[
\forall \tau_j^i : DC(i, \tau_j^i) - DC(i, \tau_j^i + 1) \leq Q_i
\]
\[
R^i_j(\tau_j^i, t) \leq Q_i + DC(i, \tau_j^i) - DC(i, \tau_j^i + 1)
\]

which leads to

\[
\forall t \in [\eta_j^i+1, \eta_{j+1}] : R^i_j(t, \tau_j^i) = pQ_i + l_i^m - \epsilon < pQ_i + l_i^m
\]

One can also focus on backlog period

\[
[\tau_k^i, \tau_{k+p}^i] \text{ i-backlog period } \Rightarrow R^i_k(\tau_k^i, \tau_{k+p}^i) \geq pQ_i - l_i^m - \epsilon > pQ_i - l_i^m
\]

Remark on equations 20 and 21. The equations 20 and 21 have the precondition \( fl(\eta_i^j) \neq i \), 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 \( \beta \) at scheduling point. Let consider \( l_i^m = Q_i \), and a \( \beta \) function defined by

\[
\beta(x) = \begin{cases} x - Q_i & \text{if } x < Q_i \\ 2x - Q_i & \text{if } x \geq Q_i \end{cases}
\]

Then, consider the case when, at time 0, two frames of the size \( l_i^m \) of the first flow are put in the first queue. Assume that the server output is exactly \( \beta \). At first iteration, \( DC[1] = 0 \), and \( DC(\tau_1^i) = 0 \). The first frame is send from time 0 to \( Q_i \), \( \tau_0^1 = 0 \), \( \eta_0^1 = Q_i \), and the deficit counter is decremented by the size of the send frame \( i.e. Q_1 \). Then, a second iteration begins, with \( DC(1, \tau_2^1) = 0 \). 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^i(\tau_0^1, \eta_0^1) = \frac{1}{2} Q_i \leq Q_i +
DC(τ^4_k) − DC(τ^1_k). This could have been solved introducing a notion of successive actions at the same instant, but the condition fl(n_j) ≠ 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 Lemma 4.1 and Theorem 4.2. Proofs are also given here for ease of reading.

Proof of Eq. 17. The DC[i] variable is initialised to 0, and decremented in one line only in the algorithm. Before decrementing the variable DC[i] by size(head(i)), it is checked that size(head(i)) ≤ DC[i], which ensures DC[i] ≥ 0.

Conversely, DC[i] is incremented only if the queue is not empty, and, in this case, while DC[i] ≥ size(head(i)), it is decremented. When the loop ends, DC[i] < size(head(i)) ≤ l^m_i, which yields Eq. 17.

Proofing bound DC[i] ≤ l^m_i − ε needs an intermediate result: DC[i] is always a multiple of ε.

The above statement is true at reset (lines [14]), and preserved when incrementing the variable by Q_i (DC[i] ← DC[i] + Q_i) — since Q_i is also a multiple of ε, and when decrementing the variable (DC[i] ← DC[i] − size(head(i))) — since packet sizes are multiple of ε. Then, from DC[i] < l^m_i, and DC[i] and l^m_i are multiple of ε, it follows that DC[i] ≤ l^m_i − ε.

Proof of Eq. 19. DC[i] is incremented only once per iteration, by quantum Q_i.

Proof of Eq. 20. Let us introduce DC'(i, τ^1_k) as the value of DC[i] between lines [13] and [14] (DC'(i, τ^1_k) ≥ 0). Since the DC[i] variable is decremented by the amount of sent data, it follows that:

\[ R_i'(τ^1_j, τ^1_k) = Q_i + DC(i, τ^1_j) - DC'(i, τ^1_j) \] (24)

And DC(i, τ^1_j + 1) = DC'(i, τ^1_j) or DC(i, τ^1_j + 1) = 0 depending on test at line [14]. Then, DC'(i, τ^1_j + 1) ≤ DC'(i, τ^1_j).

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(·, ·) values.

\[ R_i'(τ^1_j, t) = \sum_{k=0}^{p-1} R_i'(τ^1_{j+k}, τ^1_{j+k+1}) \]
\[ \leq \sum_{k=0}^{p-1} (Q_i + DC(i, τ^1_{j+k}) - DC(i, τ^1_{j+k-1})) \]
\[ = (\sum_{k=0}^{p-1} Q_i) + DC(i, τ^1_j) - DC(i, τ^1_{j+k-1}) \]
\[ \leq pQ_i + l^m_i - ε \]

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, τ^1_j) = DC(i, τ^1_j + 1). Then, sum \( \sum_{k=0}^{p-1} R_i'(τ^1_j, τ^1_{j+k}) = pQ_i + DC(i, τ^1_j) - DC'(i, τ^1_{j+k-1}) \), 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^1_k be the start of the next period of service \( (τ^1_k = \min(τ^1_m | τ^1_m ≥ s)) \). Then, each other flow has been served at most \( p + 1 \) times between s and τ^1_{k+p}.

\[ \forall j \neq i, \{ \tau^1_m \mid s ≤ \tau^1_m ≤ \tau^1_{k+p} \} \leq p + 1 \] (25)

Proof: Since the loop is always executed in the same order, and since \([s, τ^1_{k+p}]\) is a backlog period for R_i, the condition at line [23] is always true. Then, each flow is selected at most once in \([s, τ^1_k]\), they are p iterations of the loop between \( τ^1_k \) and \( τ^1_{k+p} \) 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 ≤ 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^1_i, as defined in Def. 1

Fig. 6. Illustration of fl(x), τ^1_j, η^i_j
Let \( \tau^i_k \) 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 \( p = 0 \)).

\[
\tau^i_k \overset{\text{def}}{=} \min \{ \tau^i_m \mid \tau^i_m \geq s \}
\]

\[
\mathcal{P} \overset{\text{def}}{=} \{ \tau^i_m \mid s \leq \tau^i_m, \tau^i_{m+1} < t, f_{im} = i \}
\]

\[
p = |\mathcal{P}|
\]

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, \( \mathcal{P} = \{ \tau^i_k, \ldots, \tau^i_{k+p-1} \} \). Since \( \tau^i_k \geq s \), and \( R'^i \in \mathcal{F} \) is non decreasing, we have

\[
R_i'(\tau^i_k) \geq R_i'(s)
\]  

(26)

Let us consider \( \tau^i_{k+p} \).

If \( t \) is within a service opportunity of \( i \) (\( f(t) = i \)), then \( \tau^i_{k+p} \leq t \). (\( f(t) = i \) implies that there exists \( k' \) such that \( \tau^i_{k'} \leq t < \eta^i_{k'}, \) then \( \tau^i_{k'} \not\in \mathcal{P} \) and then \( k' > k + p - 1 \), i.e. \( k \leq k' \) and since \( \tau^i \) is increasing, \( t \geq \tau^i_{k+p} \).

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

\[
R_i'(\tau^i_{k+p}) \leq R_i'(t)
\]  

(27)

Then, \( R_i'(s, t) = R_i'(t) - R_i'(s) \geq R_i'(\tau^i_{k+p}) - R_i'(\tau^i_k) \geq pQ_i - l^m - \epsilon \) from eq.22 (the backlog hypothesis \( [\tau^i_k, \tau^i_{k+p}] \) comes from the fact that \( [\tau^i_k, \tau^i_{k+p}] \subset [s, t] \) and \( [s, t] \) is a backlogged period).

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

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

\[
R_i'(s, t) \geq pQ_i - l^m - \epsilon
\]  

(28)

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

Sub-goal 2. \( \frac{\beta(t-s)-(R_i'(s,t)+\sum_j \tau_j^i(t)))}{\sum_j Q_j} \leq p + 1 \)

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

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

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

\[
\beta(t-s) \leq R_i'(t) - R_i'(s) + \sum_{j \neq i} R_j'(\tau^i_{k+p}) - R_j'(s)
\]

By Lemma 3, there are at most \( p + 1 \) cycles for each \( R_j \) between \( s \) and \( R_j'(\tau^i_{k-1}) \), i.e. \( R_j'(\tau^i_{k+p}) - R_j'(s) \leq (p + 1)Q_j + l^m_j - \epsilon \) (using eq.27).