Stability and performance bounds in cyclic networks using network calculus

Anne Bouillard[1]*

Nokia Bell Labs France, 91620 Nozay, France
anne.bouillard@nokia-bell-labs.com
[0000−0002−3345−4653]

Abstract. With the development of real-time systems and of new wireless communication technologies having strong requirements on latencies and reliability, there is a need to compute deterministic performance bounds in networks. This paper focuses on the performance guarantees and stability conditions of networks with cyclic dependencies in the network calculus framework.
Two kinds of techniques exist: backlog-based techniques compute backlog bounds for each server, and obtain good stability bounds to the detriment of the performance bounds; flow-based techniques compute performance bounds for each flow and obtain better performance bounds for low bandwidth usage, but poor stability conditions.
In this article, we propose a unified framework that combines the two techniques and improve at the same time stability conditions and performance bounds in the classical linear model. To do this, we first propose an algorithm that computes tight backlog bounds in trees for a set of flows at a server, and then develop a linear program based on this algorithm that computes performance bounds for cyclic networks. An implementation of these algorithms is provided in the Python package `NCBounds` and is used for numerical experiments showing the efficiency of the approach.

Keywords: Network calculus · cyclic networks · linear programming.

1 Introduction

New wireless communication technologies (5G) aim at providing deterministic services, with strong requirements on buffer occupancy, latency and reliability. For example, Time-Sensitive Networks (TSN) is part of the 802.1 working group [27], whose potential application to industrial and automotive networks. Critical embedded systems also become more and more complex and it becomes a necessity to compute accurate worst-case performance guarantees.

Network Calculus is a (min,plus)-based theory that computes global performance bounds from a local description of the network. These performances are the maximum backlog at a server of end-to-end delay of a flow. Examples of applications are switched network [16], Video-on-Demand [20]... It has been very

* Anne Bouillard is part of the LINCS (www.lincs.fr).

useful for analysis large embedded networks such as AFDX (Avionics Full Duplex) [13], and more recently, it has been used to model the behavior of TSN [21].

In most applications, such as AFDX, only feed-forward topologies are used. One reason is the difficulty of deriving good deterministic performance bounds in networks with cyclic dependencies. However, allowing cycles in networks would result in a better bandwidth usage and more flexible communications [1]. As a consequence, there is a strong need to design efficient methods for computing precise deterministic performance bounds in cyclic networks, which is the aim of this paper.

State of the art. Recent works have focused on computing tight performance bounds in feed-forward networks. It is proved in [10] that the problem is NP-hard for general feed-forward topologies. Bondorf *et al.* propose in [5, 6] an approximation scheme based on finding a *good* decomposition of the problem and compute performance bounds on this decomposition, while in [17], Geyer and Bondorf use Recurrent Neural Networks to find this decomposition in tree networks.

A lot of efforts has been put on the analysis of sub-classes of networks, namely sink-trees and tandems. Schmitt *et al.* introduce in [24] the concept of *Pay multiplexing only once* (PMOO) showing at the same time the difficulties in computing tight performance bounds in tandem networks, and exhibiting tight bounds in sink-trees. In [10, 12], it is proved that computing tight performance bounds in tandem networks can be done in polynomial time.

The stability of a network is still an open problem in network calculus. The most classical method for computing performance guarantees in cyclic networks is to use the *fix-point* or *time-stopping* method first presented by Cruz in [15]. A sufficient condition for stability is obtained as the existence of a fix point in an equation derived from the network description. This technique has recently been applied by Amari and Mifdaoui in [2] to ring networks, using the *multidimensional convolution* for PMOO in tandem networks first developed in [9].

Another classical result is the stability of the ring for any bandwidth usage under 100%, which is proved by Tassiulas and Georgiadis in [26] for work-conserving links and generalized in [18]. The stability condition in [2] is only 50% of the bandwidth usage in the uniform ring, performance in [2] are better in small bandwidth usages.

Other research directions focus on the FIFO policy. Rizzo and Le Boudec find sufficient condition for the stability in FIFO networks in [23]; Andrews shows in [3, 4] that the FIFO policy can be unstable at arbitrary small utilization rates.

Finally, some techniques have also been introduced to stabilize networks: Starobinski *et al.* propose in [25, 22] the *turn-prohibition* method that breaks the cyclic dependencies by forbidding some paths of length 2. This ensures both stability and connectivity. Another solution is to add regulators, named shapers, after each server. In [19], Le Boudec shows that introducing shapers allows the control of the worst-case performances.

Contributions. In this paper, we study the problem of stability in networks with cyclic dependencies in the network calculus formalism, unifying the approaches of the time-stopping method and of the backlog-based method of [26,

18]. As in most of the above references, we restrict ourselves to the linear model: when arrival curves are token-bucket and the service curves rate-latency. This approach includes several steps:

1. generalizing the recent algorithm of [12] that computes exact worst-case delays in a tandem network. It now enables to compute the worst-case backlog of a server for any subset of flows crossing that server in tree networks. As a matter of fact, the algorithm in [12] can be deduced from this new algorithm, while the reverse is not true;
2. improving the time-stopping technique. Performance bounds are computed as the solution of a linear program, allowing more expressiveness that the fixpoint method. This improvement combines with backlogged-based method, and in particular, the stability of the ring is proven without the additional technical assumption used in [18];
3. providing the Python package `NCBounds` [11] that contains this algorithm, some variants and state-of-the-art techniques.

The rest of the paper is organized as follows: in Section 2, we recall the network calculus basics. Then in Section 3, we give our algorithm that computes exact worst-case backlog in tree networks. Next in Section 4, we present a linear program to compute sufficient conditions for the stability of networks and prove the stability of some networks. Finally, we compare them through numerical experiments in Section 5.

2 Network calculus framework

In this section, we present the necessary material needed for the rest of the paper. A more complete presentation of the network calculus framework can be found in the reference books [8, 14, 18]. We use the notation $\mathbb{N}_n$ for $\{1, \dots, n\}$.

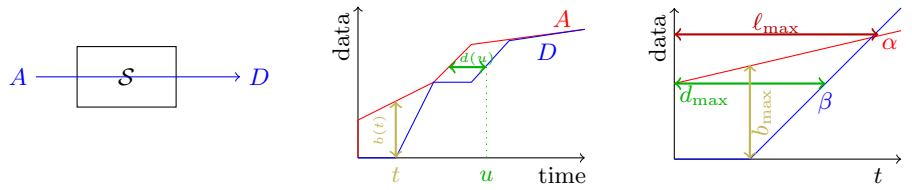
2.1 Data flows and server

Data processes and arrival curves. Flows of data are represented by cumulative processes. More precisely, if A represents a flow at a certain point in the network, $A(t)$ is the amount of data of that flow that crosses that point in the time interval $[0, t)$, with the convention $A(0) = 0$. The processes are non-decreasing and left-continuous. We denote by $\mathcal{F}$ the set of such functions.

A flow A is constrained by the arrival curve α, or is α-constrained, if

$$\forall s, t \in \mathbb{R}_+ \text{ with } s \leq t, \quad A(t) - A(s) \leq \alpha(t - s).$$

In the following we will consider *leaky-bucket* functions: $\gamma_{b,r} : 0 \mapsto 0;\ t \mapsto b + rt$, if $t > 0$. The *burst* b can be interpreted as the maximal amount of data that can arrive simultaneously and the *arrival rate* r as a maximal long-term arrival rate of data.



(a) Server model (b) Processes (c) Performances

Fig. 1: Server model and worst-case performances.

Servers and service curves. An n-server $\mathcal{S} \subseteq \mathcal{F}^n \times \mathcal{F}^n$ (illustrated for $n = 1$ in Figure 1a) is a relation between n arrival processes $(A_i)_{i=1}^n$ and n departure processes $(D_i)_{i=1}^n$ such that $A_i \geq D_i$ for all $i \in \mathbb{N}_n$. The latter inequality models the causality of the system (no data is created inside the system).

The role of a service curve is to constrain the relation between the inputs of a server and its outputs. Several types of service curves have been defined in the literature (see [18]), and we here only focus on the strict service curve. Intuitively, a strict service curve gives the minimum amount of service provided to the arrival processes provided the system is not empty. More formally, an interval I is a *backlogged period* for $(A, D) \in \mathcal{F} \times \mathcal{F}$ if $\forall u \in I$, $A(u) > D(u)$.

We say that $\beta \in \mathcal{F}$ is a *strict service curve* for 1-server $\mathcal{S}$ if

$$\forall (A, D) \in \mathcal{S}, \quad A \geq D \quad \text{and } \forall \text{ bckl. per. } (s, t], \ D(t) - D(s) \geq \beta(t - s). \tag{1}$$

In the following we will use the *rate-latency* service curves: $\beta_{R,T} : t \mapsto R(t - T)_+$, where T is the *latency* until the server has to become active and R is its minimal *service rate* after this latency.

A n-server $\mathcal{S}$ offers a strict service curve β if, seen as a 1-server, $\mathcal{S}$ offers β to $\sum_{i=1}^m A_i$ and $\sum_{i=1}^m D_i$ for all $((A_i), (D_i)) \in \mathcal{S}$. We call the flow with arrival process $\sum_{i=1}^m A_i$ the aggregate flow of flows $1, \ldots, n$. We assume no knowledge about the service policy in this system (except that it is FIFO per flow).

2.2 Performance guarantees in a server

Backlog and delay. Let $\mathcal{S}$ be a 1-server and $(A, D) \in \mathcal{S}$. The backlog of that server at time t is $b(t) = A(t) - D(t)$. The worst-case backlog is then $b_{\max} = \sup_{t \geq 0} b(t)$.

We denote $b_{\max}(\alpha, \beta)$ the maximum backlog that can be obtained for an α-constrained flow crossing a server with strict service curve β. For example, we have $b_{\max}(\gamma_{b,r}, \beta_{R,T}) = b + rT$ if $r \leq R$.

The delay of data exiting at time t is $d(t) = \sup\{d \geq 0 \mid A(t - d) - D(t)\}$. The worst-case delay is then $d_{\max} = \sup_{t \geq 0} d(t)$.

We denote $d_{\max}(\alpha, \beta)$ the maximum delay that can be obtained for an α-constrained flow crossing a server with strict service curve β. For example, we have $d_{\max}(\gamma_{b,r}, \beta_{R,T}) = T + \frac{T}{R}$ if $r < R$.

Backlog and delay are illustrated on Figures 1b and 1c.

Stability Our main interest is the network stability.

Definition 1 (Server stability). *Consider a server offering a strict service curve β and crossed by an α-constrained flow. This server is said* stable *if its backlogged periods are bounded.*

If the service curve is rate-latency $\beta_{R,T}$ and the arrival curve leaky-bucket $\gamma_{b,r}$, then a server is stable if and only if $R > r$, as the length of a backlogged period is $\sup\{t > 0 | \alpha(t) > \beta(t)\}$ and is $\ell_{\max}(\gamma_{b,r}, \beta_{R,T}) := \frac{b+RT}{R-r}$ in that case.

This definition involves r and R only. The stability is insensitive to b and T, that only influence the server's performance. This also explains why we restrict in this paper to these types of curves: if a more general arrival (resp. service) curve can be and upper bounded by some token-bucket (resp. rate-latency) functions with the same rate, then the stability sufficient conditions that we compute in this paper are not impacted by the approximation by token bucket and rate latency functions.

2.3 Network model

Consider a network composed of n servers numbered from 1 to n and crossed by m flows named $f_1, \ldots, f_m$, such that

- each server j guarantees a strict service curve β_j;
- each flow f_i is α_i-constrained and circulates along an acyclic path $\pi_i = \langle \pi_i(1), \ldots, \pi_i(\ell_i) \rangle$ of length ℓ_i.

We call the model *linear* when arrival curves are leaky-bucket and the service curve rate-latency.

For a server j, we define $\mathrm{Fl}(j) = \{i \mid \exists \ell,\ \pi_i(\ell) = j\}$ the set of indices of the flows crossing server j.

We denote by $\mathcal{N}$ this network. The induced graph $G_{\mathcal{N}} = (\mathbb{N}_n, \mathbb{A})$ is the directed graph whose vertices are the servers and the set of arcs is

$$\mathbb{A} = \{(\pi_i(k), \pi_i(k+1)) \mid i \in \mathbb{N}_m, k \in \mathbb{N}_{\ell_i - 1}\}.$$

As we will focus on the performances in server n, we can assume without loss of generality that the network is connected and has a unique final strictly connected component.

Tree networks If the induced graph $G_{\mathcal{N}}$ has out-degree 1 for each vertex except node n is 1 and out-degree 0 for vertex n, then $\mathcal{N}$ is called a *tree*. In that case, we denote by $j^\bullet$ the unique successor of server j and assume that $j < j^\bullet$, with the convention $n^\bullet = n + 1$. The set of predecessors of a vertex is ${}^\bullet j = \{k \mid k^\bullet = j\}$. There exists at most one path between two vertices j and k, denoted $j \rightsquigarrow k$, and if there exists such a path, ${}^{\bullet j}k$ is the predecessor of k on this path. Figure 2 illustrates the notations of the network model.

Finally we can extend the notion of stability to networks.

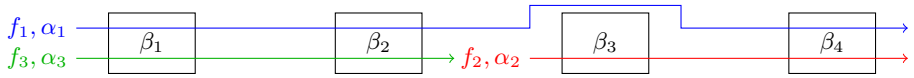


Fig. 2: Example of a tree-networks with 3 flows and 4 servers. To illustrate the notations, we have $\pi(1) = \langle 1, 2, 4\rangle$, $\mathrm{Fl}(4) = \{1, 2\}$, $4^\bullet = \{2, 3\}$, $4^{\bullet_1} = 2$, and $1 \rightsquigarrow 4 = \langle 1, 2, 4\rangle$.

Definition 2 (Local stability). *A network $\mathcal{N}$ is* locally stable *if all its servers are stable using the initial arrival curves:*

$$\forall j \in \mathbb{N}_n, \quad \ell_{\max}(\sum_{i \in \mathrm{Fl}(j)} \alpha_i, \beta_j) < \infty.$$

Definition 3 (Global stability). *A network is* globally stable *if the backlogged periods of each server are uniformly bounded.*

It is well-known that if a network is globally stable, then it is locally stable. The converse is true for feed-forward networks, but not in general.

3 Worst-case backlog in trees

In this section, we focus on tree networks and give an algorithm to compute exact worst-case backlog in the linear model. Then we compare with the existing approaches.

3.1 Algorithm

The algorithm presented in this paragraph is a generalization of the one given in [12] with the following differences:

1. our algorithm computes a worst-case backlog at a server;
2. it can be applied to compute the worst-case backlog at a server for any set of flows crossing this server;
3. it is valid for any tree topology.

The two algorithms and their proof are based on the same ideas, and due to the space limitation, we do not present the complete proof here.

Theorem 1. *Consider a stable tree network with n servers offering a rate-latency strict service curves β_{R_j,T_j}, and m flows with leaky-bucket arrival curves γ_{b_i,r_i}. Let I be a subset of flows crossing server n. Then there exists $(\rho_j)_{j\in\mathbb{N}_n}$ and $(\varphi_i)_{i\in\mathbb{N}_n}$ such that the worst-case backlog at server n for flows in I is*

$$B = \sum_{j=1}^{n} \rho_j T_j + \sum_{i=1}^{m} \varphi_i b_i, \tag{2}$$

where the coefficients ρ_j and φ_i depend only on r_i and R_j and are computed by Algorithm 1. This algorithm runs in time $O(n^2 + m)$.

If there is only one flow for each possible source/destination pair, then $m \leq n^2/2$ and the algorithm runs is $O(n^2)$.

We call $I \subseteq \mathbb{N}_m$ the set of flows of interest, and use the following additional notations:

- $r_j^k = \sum_{i \in \mathrm{Fl}(j) \setminus I, \pi_i(\ell_i)=k} r_i$ is the arrival rate at server j for all flows ending at server k and crossing server j that are not of interest;
- $r_j^* = \sum_{i \in I \cap \mathrm{Fl}(j)} r_i$ is the arrival rate of the flows of interest that cross server j.

Algorithm 1: Worst-case backlog algorithm

begin
  $\xi_n^n \leftarrow r_n^*/R_n - r_n^n$;
  $Q = \mathtt{queue}({}^\bullet n)$;
  while $Q \neq \emptyset$ **do**
    $j = Q[0]$;
    $k \leftarrow n$;
    while $\xi_{j^\bullet}^k > (r_j^* + \sum_{\ell \in k^\bullet \rightsquigarrow n} \xi_{j^\bullet}^\ell r_j^\ell)/(R_j - \sum_{\ell \in j \rightsquigarrow k} r_j^\ell)$ **do**
      $\xi_j^k \leftarrow \xi_{j^\bullet}^k$;
      $k \leftarrow {}^{\bullet_j}k$;
    for ℓ **from** j **to** k **do** $\xi_j^\ell \leftarrow (r_j^* + \sum_{\ell' \in k^\bullet \rightsquigarrow n} \xi_{j^\bullet}^{\ell'} r_j^{\ell'})/(R_j - \sum_{\ell \in j \rightsquigarrow k} r_j^{\ell'})$;
    $Q \leftarrow \mathtt{enqueue}(\mathtt{dequeue}(Q, j), {}^\bullet j)$;
  for j **from** 1 **to** n **do** $\rho_j \leftarrow r_j^* + \sum_{\ell \in j \rightsquigarrow n} \xi_j^\ell r_j^\ell$;
  for i **from** 1 **to** m **do**
    if $i \in I$ **then** $\varphi_i \leftarrow 1$;
    else $\varphi_i \leftarrow \xi_{\pi_i(1)}^{\pi_i(\ell_i)}$;

Proof (Sketch). The proof is in two steps. First we show that there is a worst-case trajectory (*i.e.* a set of arrival and departure processes satisfying all arrival and service curve constraints and reaching the worst-case backlog at a time denoted $t_{n+1} = t_{n^\bullet}$) satisfying the following properties:

(P_1) The service policy is SDF (shortest-to-destination-first): priority is given to flows that stop at the server with the smallest number.

(P_2) For each server j, there is a unique backlogged period $(t_j, t_{j^\bullet}]$, where the service offered is as small as possible. After and before this backlogged period, the server transmits data instantaneously.

(P_3) The arrival cumulative process of flow f_i entering the system at server j (*i.e.* $\pi_i(1) = j$) is maximal from t_j, the start of the backlogged period of server j (it is $\alpha_i(t - t_j)$ for all $t > t_j$ and 0 otherwise. Intuitively, the backlog does not increase if the cross-traffic is delayed).

(P_4) Data from the flows of interest in I have the lowest priority and are instantaneously served by server j at time $t_{j^\bullet}$ (if they cross server j) and are all in server n at time t_{n+1}.

The second step is to find a worst-case trajectory. The first step allows us to reduce the space of the trajectories, and in fact only the dates $(t_j)_{j\in\mathbb{N}_n}$ remain to optimize. This is done by a backward induction on the servers. The choice of date t_j is equivalent to choosing which flows (hence the quantity of data) will be transmitted instantaneously by server j to its successor, so that the backlog in the final server is maximized. □

The worst-case delay of a flow (main results of [12]) can be deduced from the worst-case backlog when I is reduced to this flow.

Corollary 1. *Suppose that flow* 1 *crosses server* n*. Then the worst-case delay of flow* 1 *starting at server* j *and ending at server* n *is*

$$\Delta = \frac{B - b_1}{r_1} + \frac{\xi_j^n b_1}{r_1},$$

where B *and* ξ_j^n *are the worst-case backlog and coefficient obtained from Algorithm 1 when* $I = \{1\}$*.*

Interpretation of ξ*.* The parameter ξ_j^ℓ can be interpreted as the contribution of data crossing server j and exiting at server ℓ contribute to increase the backlog of the flows of interest at server n. For example, consider server n, and suppose that for the flows not of interest, the backlog transmitted from server $n-1$ to server n and from flows entering at server n at time t_{n-1} is b. Denote by r_n^n the arrival rate of these flows. These data have the highest priority. All data from these flows are served after a time $t = t_n - t_{n-1} = T_n + \frac{b+r_n^n T_n}{R_n - r_n^n}$ (we have $b + r_n^n t = R_n(t-T)_+$). During this time, data created by the flows of interest, at rate r_n^*, can be divided in two parts: $r_n^* T_n$ induced by server n's latency and $r_n^* \frac{b+r_n^n T_n}{R_n - r_n^n}$ induced by the interference of the other flows, which explains the equality $\xi_n^n = \frac{r_n^*}{R_n - r_n^n}$. The interpretation is similar for the other servers. The complexity of the formula and the comparison of line 7 corresponds to searching the downstream bottleneck.

3.2 Backlog and arrival curves for aggregation of flows

In this paragraph, we show how to use our algorithm to compute arrival curves for the aggregation of flows, and how this can be used to improve the performances bounds computed: Theorem 2 is the base for our backlogged based approach and Theorem 3 improves the performance bounds by combining flows and backlog.

Theorem 2. *With the same notations and assumptions as in Theorem 1, the arrival curve of the departure functions from server* n *for flows in* I *is* $\gamma_{B,\sum_{i\in I} r_i}$*.*

This theorem is a generalization to networks and for several flows of the classical result that characterizes the arrival curve of a departure process (see [8, Theorem 5.3] for example).

Consider a tree network as above and the following assumptions:

- $I = \{f_1, \ldots, f_k\}$ is a set of flows all entering the network at server j, that have respective arrival curves γ_{b_i,r_i}. Data of each flow f_i can arrive at rate at least r_i independently of the other flows;
- the arrival process of the aggregation of the flows in I is constrained by the arrival curve $\gamma_{b,r}$, with $r = \sum_{i \in I} r_i$;
- B is a performance of the tree network (backlog at its root or delay of a flow), and $(\varphi_i)_{i \in I}$ are the computed with Algorithm 1.

In the rest of the paper, the assumption that each flow f_i can arrive at rate r_i independently of the other flows is satisfied, because the servers can serve data instantaneously and any service policy is possible.

Theorem 3. *With the notations and assumptions above,*

$$B \leq \sup\{\sum_{i=1}^{k} \varphi_i x_i + C \mid x_i \leq b_i \text{ and } \sum_{i \in I} x_i \leq b\},$$

where C is a constant including the contribution of all flows not in I and of all latencies.

Proof. From property (P_3), the arrival processes A_i of $f_i \in I$ are maximal from time t_j, the start of backlogged period of server j. Then data from flow f_i can arrive at rate r_i at least. As this is also the maximal long-term arrival rate, there exists $x_i \leq b_i$ such that $A_i(t) - A_i(t_j) \leq x_i + r_i(t - t_j)$, and that $\limsup_t A_i(t) - A_i(t_j) - r_i(t - t_j) \geq x_i$. We then have the inequality $B \leq \sum_{i=1}^{k} \varphi_i x_i + C$.

But we also have

$$\sum_{i \in I} A_i(t) - A_i(t_j) \leq b + \sum_{i \in I} r_i(t - t_j),$$

so $\sum_{i \in I} x_i \leq b$. As a consequence, the performance of the system can be bounded by $B \leq \sup\{\sum_{i=1}^{k} \varphi_i x_i + C \mid x_i \leq b_i \text{ and } \sum_{i \in I} x_i \leq b\}$. □

3.3 Examples and comparison with the state of the art

We compare our performance bounds with the state of the art for two models that have been widely studied in the literature: the sink-tree from [24, 7] and the tandem networks (PMOO technique from [9, 2]).

Sink-trees are tree topologies where the destination of every flow is the root (node n). In this special case, each iteration of the external loop (lines 5-11) can be performed in constant time (there is only one test to perform). Moreover, the number of flows is at most the number of servers. As a consequence, our algorithm can be performed in $O(n)$.

To compute the maximum backlog at the root, every flow is a flow of interest, so $\varphi_i = 1$ for each flow i and $\rho_j = r_j^* = \sum_{i \in \mathrm{Fl}(j)} r_i$. It is easy to check that the formula is the same as in [7, Theorem 14].

Fig. 3: Left: sink-tree, Right: tandem.

	ξ_2^2	ξ_1^2	ξ_1^1	Algo 1	[9, Cor. 25]	[7, Th. 18, 15]	[24, Sec. V]
f_1	$\frac{r}{2R-r}$	$\frac{r}{R}$	$\frac{2r}{2R-r}$	$T + \frac{b}{R} + \frac{b+rT}{2R-r}$	$2T + \frac{2b+rT}{R}$	$2T + \frac{2b+rT}{R}$	$T + \frac{b}{R} + \frac{b+rT}{2R-r}$
f_2	$\frac{r}{2R-r}$	$\frac{r}{2R-r}$	$\frac{r^2}{(R-r)(2R-r)}$	$\frac{2b+(2R+r)T}{2R-r}$	∞	$\frac{2b+(2R+r)T}{2R-r}$	∞
f_3	$\frac{r}{R-r}$	$\frac{r}{R-r}$	$\frac{r}{R-r}$	$\frac{2(TR+b)}{R-r}$	$\frac{2(TR+b)}{R-r}$	∞	∞
f_4	$\frac{r}{R-r}$	$\frac{r}{R-r}$	$(\frac{r}{R-r})^2$	$\frac{R(TR+b)}{(R-r)^2} + \frac{b}{R-r}$	∞	∞	∞

Table 1: Comparison of delay bounds.

Table 1 shows the values of ξ_i^j for the two examples of Figures 3 and flows $f_1, \ldots, f_4$, where all flows are constrained by the arrival curve $\gamma_{b,r}$, and the comparison of the worst-case delay obtained in this case against other techniques. We write "∞" when no specific mean is provided to compute de performance.

We see that our algorithm is strictly more general than the study of some specific topologies in several ways: of course we can handle tree topologies that strictly contains tandem and sink-tree networks. But, it can also compute performances for flows of interest that do not cross *all* the servers, which only [7] does for sink-trees.

The PMOO bound gives the tight performance in the tandem network, but not in the sink-tree. The tandem here is quite specific, as all the coefficients ξ_i^j are the same. Our guess is that the PMOO formula is an upper bound of the bounds found by our algorithm, obtained by replacing ξ_i^j by $\max_{i,j} \xi_{i,j}$. So equality occurs when all the coefficients are the same.

4 Computing performances in cyclic networks

In this section, we generalize the fix-point method, also known as time-stopping, that is used to compute the worst-case performance bounds in networks with cyclic dependencies. The principle of this method is to *split* the paths of the flows into sub-paths in order to obtain an acyclic network, for which it is possible to compute performance bounds, in particular output arrival curves at the end of each sub-path. Then to retrieve performance guarantees of the original network, a fix-point on the arrival curves is computed: an output arrival curve at the end of a sub-path is the arrival curve at the start of the next sub-path. Details on this approach can be found in [18, 8].

In this section we generalize this approach to take advantage of the results derived in Section 3.2. In Section 4.1, we present a linear program computing worst-case performance bounds and in Section 4.2, we show the stability of the ring and slightly more general networks.

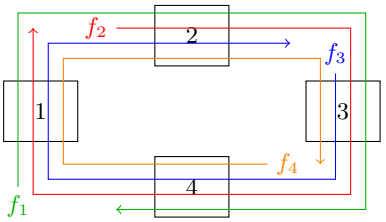


Fig. 4: Ring network with $n = 4$.

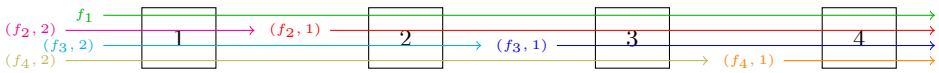


Fig. 5: Uniform tandem network with $n = 4$.

4.1 Decomposition in trees and linear program

We apply the time-stopping method when decomposing the network into trees. The graph of the network $G_{\mathcal{N}}$ can be transformed into a forest by removing a set of arcs $\mathbb{A}^r$. In the network, each flow f_i that traverse removed arcs is *split* into several flows $(f_i, 1), (f_i, 2), \ldots, (f_i, m_i)$ of respective paths in $(\mathbb{N}_i, \mathbb{A} - \mathbb{A}^r)$, $\pi_{i,1} = \langle \pi_i(1), \ldots, \pi_i(k_1) \rangle$, $\pi_{i,2} = \langle \pi_i(k_1 + 1), \ldots, \pi_i(k_2) \rangle$,..., $\pi_{i,m_i} = \langle \pi_i(k_{m_i} + 1), \ldots, \pi_i(\ell_i) \rangle$, where $(\pi_i(k_j + 1), \pi_i(k_j)) \in \mathbb{A}^r$. Figure 4 represents a ring of length 4. One can choose $\mathbb{A}^r = (4, 1)$, and flows f_2, f_3, f_4 can be decomposed into $(f_i, 1)$ and $(f_i, 2)$, $i \in \{2, 3, 4\}$, as depicted in Figure 5.

Consider $\mathcal{N}^{\text{FF}}$, the feed-forward network obtained after this transformation and let us focus on two kinds of arrival curves:

- the arrival curve of each flow (f_i, k), denoted $\alpha_{i,k}$;
- the arrival curve of the aggregation of flows crossing each removed arc of $\mathbb{A}^r$, and denoted λ_a, and that we call arrival curve of arc a.

If all these arrival curves are known and finite, then the performance bounds can be computed for every flow.

In the original network $\mathcal{N}$, all these arrival curves are *a priori* not known, except $\alpha_{i,1} = \alpha_i$, but we can write equations where these arrival curves are the unknowns. More precisely, suppose that the network is stable and denote $\alpha_{i,k}$ the *best* arrival curve (that is, the minimum arrival curves among all the possible arrival curves) of flow k at server $p_{i,k}(1)$ and λ_a the *best* arrival curve of arc a. From Theorems 2 and 3, there exist functions $F_{i,k}$ and F_a such that

$$\alpha_{i,k} \leq F_{i,k}((\alpha_s)_{s \in S}, (\lambda_a)_{a \in \mathbb{A}^r}) \quad \text{and} \quad \lambda_a \leq F_{i,k}((\alpha_s)_{s \in S}, (\lambda_a)_{a \in \mathbb{A}^r}),$$

with $S = \{(i, k), i \in \mathbb{N}_m, 1 < k \leq m_i\}$.

Using vector notation, with $(\boldsymbol{\alpha}, \boldsymbol{\lambda}) = ((\alpha_s)_{s \in S}, (\lambda_a)_{a \in \mathbb{A}^r})$ and $\mathbf{F} = (F_s)_{s \in S \cup \mathbb{A}^r}$, one can write, $(\boldsymbol{\alpha}, \boldsymbol{\lambda}) \leq \mathbf{F}(\boldsymbol{\alpha}, \boldsymbol{\lambda})$ and the following theorem holds.

Theorem 4. *Set* $\mathcal{C} = \{(\boldsymbol{\alpha}, \boldsymbol{\lambda}) \mid (\boldsymbol{\alpha}, \boldsymbol{\lambda}) \leq \mathbf{F}(\boldsymbol{\alpha}, \boldsymbol{\lambda})\}$ *and* $(\tilde{\boldsymbol{\alpha}}, \tilde{\boldsymbol{\lambda}}) = \sup\{(\boldsymbol{\alpha}, \boldsymbol{\lambda}) \mid (\boldsymbol{\alpha}, \boldsymbol{\lambda}) \in \mathcal{C}\}$. *If* $(\tilde{\boldsymbol{\alpha}}, \tilde{\boldsymbol{\lambda}})$ *is finite, then* $\mathcal{N}$ *is globally stable and for all* s, $\tilde{\alpha}_s$ *is an arrival curve for flow* s *and for all* $a \in \mathbb{A}^r$, $\tilde{\lambda}_a$ *for arc* a.

The proof of this theorem is nearly the same as that of [8, Theorem 12.1].

In the linear model, the arrival rate of each arrival curve remains the same as the original flow, and we only have to compute the burst b_s of each flow s, and B_a is the backlog at each arc in $\mathbb{A}^r$. A sufficient condition for the stability can be expressed as a linear problem. Consider $\mathcal{L}$ the following set of linear constraints:

$$\mathcal{L} = \left\{ \begin{array}{ll} b_s \leq \sum_{s' \in S} \varphi^s_{s'} x^s_{s'} + C_s, & \forall s \in S \\ B_a \leq \sum_{s' \in S} \varphi^a_{s'} x^a_{s'} + C_a, & \forall a \in \mathbb{A}^r \\ 0 \leq x^s_{s'} \leq b_{s'}, & \forall s' \in S, s \in S \cup \mathbb{A}^r \\ \sum_{s' \in a} x^s_{s'} \leq B_a, & \forall a \in \mathbb{A}^r, s \in S \cup \mathbb{A}^r \end{array} \right\},$$

where we write $(i,k) \in a$ if flows $(f_i, k-1)$ and (f_i, k) have been split by arc a and where $\varphi^s_{s'}$ are the coefficient obtained by Algorithm 1 with flow(s) of interests $s \in S \cup \mathbb{A}^r$ and C_s the contribution of the latencies.

Theorem 5. *If* $\mathcal{L}$ *is bounded, then the system is stable.*

This is a rephrasing of Theorem 4, using the linear constraints given in Theorem 3: constraints $\{b_s \leq \sum_{s' \in S} \varphi^s_{s'} x^s_{s'} + C_s, 0 \leq x^s_{s'} \leq b_{s'}, \forall s' \in S, \sum_{s' \in a} x^s_{s'} \leq B_a, \forall a \in \mathbb{A}^r\}$ represent the constraints for computing the backlog of flow s, as in Theorem 3 applied to flow s, and similarly $\{B_a \leq \sum_{s' \in S} \varphi^a_{s'} x^a_{s'} + C_a, 0 \leq x^a_{s'} \leq b_{s'}, \forall s' \in S, \sum_{s' \in a'} x^a_{s'} \leq B_{a'}, \forall a' \in \mathbb{A}^r\}$ are the constraints coming from Theorem 3 applied to computing the worst-case backlog at arc a.

A linear program to compute a performance bound of the system (worst-case delay or worst-case backlog) is then

$$\begin{array}{lll} \text{Maximize :} & \sum_{s' \in S} \varphi_s y_s + C & \\ \text{Constraints :} & \sum_{s \in a} y_s \leq B_a, & \forall a \in \mathbb{A}^r, \\ & 0 \leq y_s \leq b_s, & \forall s' \in S, \\ & \mathcal{L}. & \end{array}$$

The linear program has $O(c^2)$ variables and constraints, with $c = |S| + |\mathbb{A}^r|$, and c constraints require using the quadratic Algorithm 1, which can be costly. The next paragraph shows two ways to relax the problem to respectively $O(|S|)$ and $O(|\mathbb{A}^r|)$ variables and constraints.

Flow-based and backlog-based linear programs. The linear program we just wrote combines two techniques usually used separately for cyclic networks: the first one uses the time-stopping technique to compute the characteristics of each flow at servers it crosses, and we call them flow-based. The second one computes worst-case backlog in each server, similar to [26, 18], and we call them backlog-based.

The set $\mathcal{L}$ can be simplified into $\mathcal{L}_F$ and $\mathcal{L}_B$ to derive respectively flow-based and backlog-based bounds: it suffices to respectively keep the flow related

constraints (first and third lines of $\mathcal{L}$) for $\mathcal{L}_F$ and the arc related constraints (second and fourth lines of $\mathcal{L}$) for $\mathcal{L}_B$. Of course, larger performance bounds will be obtained, but we will see in Section 5 that these linear programs already improve the flow-based or backlog-based bounds.

As $\varphi^s_{s'}$ are non-negative, variables $x^s_{s'}$ become useless, and we finally obtain:

$$\begin{aligned}\mathcal{L}_F &= \left\{ \begin{array}{l} b_s \leq \sum_{s' \in S} \varphi^s_{s'} x^s_{s'} + C_s, \forall s \in S \\ 0 \leq x^s_{s'} \leq b_{s'} \forall s' \in S, s \in S \cup \mathbb{A}^r \end{array} \right\} \\ &= \left\{ b_s \leq \sum_{s' \in S} \varphi^s_{s'} b^s_{s'} + C_s, \forall s \in S \right\},\end{aligned}$$

and

$$\begin{aligned}\mathcal{L}_B &= \left\{ \begin{array}{l} B_a \leq \sum_{s' \in S} \varphi^a_{s'} x^a_{s'} + C_a, \forall a \in \mathbb{A}^r \\ \sum_{s' \in a} x^s_{s'} \leq B_a \forall a \in \mathbb{A}^r, s \in S \cup \mathbb{A}^r \end{array} \right\} \\ &= \left\{ B_a \leq \sum_{a' \in \mathbb{A}^r} (\max_{s' \in a'} \varphi^a_{s'}) B_{a'} + C_a, \forall a \in \mathbb{A}^r \right\}.\end{aligned}$$

4.2 Stability of the ring

Consider a ring with n servers. Its induced graph is $\mathcal{G}$ with $\mathbb{A} = \{(i, i+1), i \leq n-1\} \cup \{(n, 1)\}$. The transformation into a tree is a tandem network obtained by removing arc $(n, 1)$. Flows are decomposed in either one flow if $\pi_i(1) < \pi_i(\ell_i)$ or two flows otherwise: flow $(f_i, 1)$ has path $\langle \pi_i(1), \ldots, n \rangle$ and flow $(f_i, 2)$ has path $\langle 1, \ldots, \pi_i(\ell_i) \rangle$.

Theorem 6. *The ring is stable under local stability condition.*

Proof. The ring is stable if the set $\mathcal{L}_B$ is bounded, that is, if $\varphi^{(n,1)}_s < 1$ for all $s \in S = \{(i, 2) \mid \forall i \text{ such that } \pi_i(1) > \pi_i(\ell_i)\}$.

To compute B_a, the flows of interest are flows $(i, 1)$, so $\varphi^a_{(i,2)} = \xi_1^{\pi_i(\ell_i)}$.

Observe from Algorithm 1 how ξ^ℓ_j are computed: because of the local stability, $R_n > r^n_n + r^*_n$, so $\xi^n_n < 1$. Now assume that $\xi^k_{j^\bullet} < 1$ (lines 7-9). Either $\xi^k_j = \xi^k_{j^\bullet} < 1$, or $\xi^\ell_j = (r^*_j + \sum_{\ell' \in k^\bullet \rightsquigarrow n} \xi^{\ell'}_{j^\bullet} r^{\ell'}_j)/(R_j - \sum_{\ell \in j \rightsquigarrow k} r^{\ell'}_j) \leq (r^*_j + \sum_{\ell' \in k^\bullet \rightsquigarrow n} r^{\ell'}_j)/(R_j - \sum_{\ell \in j \rightsquigarrow k} r^{\ell'}_j) < 1$, from the local stability condition. As a consequence, for all j and ℓ, $\xi^\ell_j < 1$ and $\max_{s \in S} \varphi^{(n,1)}_s < 1$, which ends the proof. □

This result has already been proved under stronger assumptions: in [26], servers are constant-rate servers and in [18], servers have a maximal service rate. Our method is not specific to the ring topology, so we can hope to improve the stability conditions for more general topologies.

Stability of hierarchical cycles. A straightforward generalization is when the network only has disjoint cycles: each non trivial strongly connected component of the network is a ring. In this case, the stability can be established by induction: perform a topological ordering of the cycles, and compute performances at the outputs of each cycle in this topological order.

Unfortunately, our linear program is not enough to prove the stability of other classes of networks, and we will see in the next section that the stability condition established for a network composed of two rings is stronger than the local stability.

5 Numerical Evaluation

In this section, we compare our approach with the state of the art on several examples. The first one is the ring already defined. Indeed, the ring is the topology which has been studied in [2] and [26]. To demonstrate the generality of our algorithm, we also take the example of a network composed of two rings, but we can only compare this example with the most naive methods of [15, 18]. The different approaches have been implemented in the Python Package `NCBounds` [11]. Note that the implementation of the package aims at clarity rather than efficiency.

The following approaches are compared.

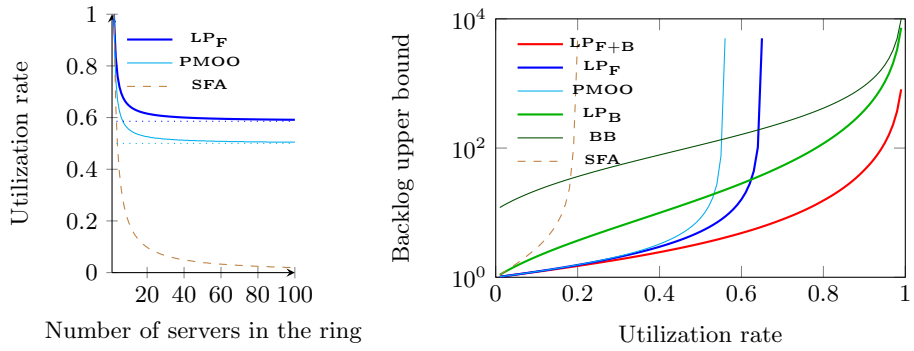
- **SFA**: the fix-point approach described in [18, Section 6.3.2] computes a fix-point in the performances of each flow at each server it crosses;
- **PMOO**: the fix-point approach described in [2] that is exemplified on the uniform ring;
- **BB**: the backlog bound of [18, Theorem 6.4.1];
- $\mathbf{LP_F}$: the flow-based linear programming approach using the $\mathcal{L}_F$ (that can be compared to **SFA** and **PMOO**);
- $\mathbf{LP_B}$: the backlog-based linear programming approach using $\mathcal{L}_B$ (that can be compared to **BB**);
- $\mathbf{LP_{F+B}}$: the linear programming approach using all constraints $\mathcal{L}$.

5.1 Uniform ring example

Consider a uniform ring network as described in Figure 4 composed of n servers and n flows of length n. Each server has a service rate $R = 100Mb.s^{-1}$ and latency of $1ms$, the maximum burst of each flow is $1Mb$. The arrival rate depends on the utilization rate $u \in [0, 1]$ and is $r = \frac{uR}{n}$.

We first compare the stability region for each method that do not stabilize the ring, namely **SFA**, **PMOO** and $\mathbf{LP_F}$. Figure 6a depicts the stability region when the number of servers varies from 2 to 100. As expected, **PMOO** provides better bounds than **SFA**, and $\mathbf{LP_F}$ improves the stability region. As conjectured in [2], the stability region with **PMOO** converges to a utilization rate of 0.5. The stability region of $\mathbf{LP_F}$ seem to converge to $2 - \sqrt{2} \simeq 0.58$, hence already providing approximately 18% improvement of the stability region over flow-based methods.

Now we fix the number of servers $n = 10$, and compare the worst-case backlogs of flow 1 at server 10 (Figure 6b). We choose backlog over delay because **BB** is more suited to this performance, and computing delays would lead to even worse performances. We observe the same stability bounds as above for the three



(a) Stability sufficient conditions.

(b) Backlog bound for 10 servers.

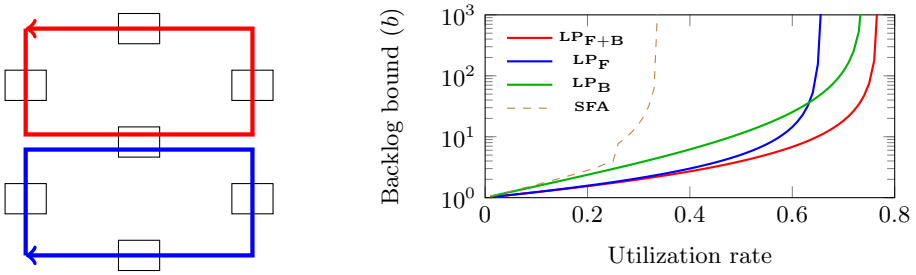
Fig. 6: Comparisons with the uniform ring.

flow-based methods. The stability of the ring is experimentally verified by the three other methods. The more constraints we add in the linear programming approach, the tighter the backlog bound, hence $\mathbf{LP_{F+B}}$ is a better bounds than $\mathbf{LP_F}$ and $\mathbf{LP_B}$. Second, $\mathbf{LP_F}$ also beats **PMOO**. This could be seen as surprising as **PMOO** computes tight bounds for uniform tandems. The difference can be explained by the more general applicability of Algorithm 1 stated in Section 3.3: exact worst-case performances can be computed for a flow that *do not cross all servers*, hence the network is decomposed into fewer elements, which induces less pessimism in the performance bounds. $\mathbf{LP_B}$ also beats **BB**. This is quite logical, as the formula uses fewer parameters that the linear program.

5.2 Two-ring example

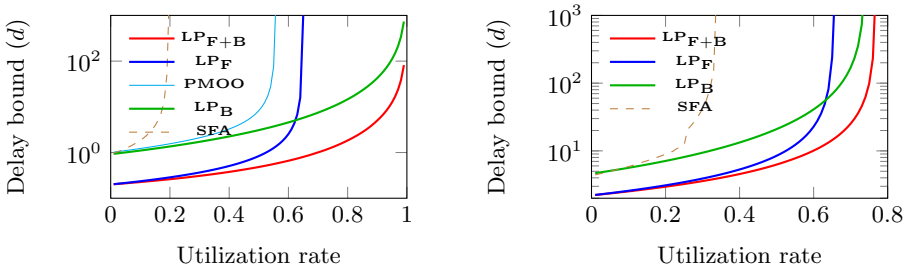
We now consider a network composed of two rings of length n, as depicted in Figure 7a with $n = 4$. Each flow has length n and circulates along one of the two rings ($2n$ flows), and the description of the servers and flows is the same as above, except that the central server has service rate $2R$. Figure 7b compares the performances obtained with the three **LP** methods against **SFA**, the only other method that can be applied to non-ring networks. We see that the stability region and the performances are improved. In this case, we do not obtain the stability (which is an open issue), but $\mathbf{LP_{F+B}}$ strictly improve the stability region ($u \leq 0.76$) of both $\mathbf{LP_F}$ ($u \leq 0.75$) and $\mathbf{LP_B}$ ($u \leq 0.73$), while the two latter methods do not compare performance-wise.

Backlog *vs* delay. Finally, Figures 8a and 8b depict the delays of flow 1 in the same experimental settings as above. From Corollary 1, the delay is obtained from the backlog by a linear transformation. Consequently, the comparisons remain similar. Still, one can notice that the backlog-based bound is not very good at low utilization rate.



(a) Two-ring network. (b) Backlog bound for $n = 5$.

Fig. 7: Comparisons with the two-ring network.



(a) Uniform ring network with $n = 10$. (b) Two-ring network with $n = 5$.

Fig. 8: Delay bounds.

6 Conclusion

In this article, we gave an algorithm to compute tight worst-case performances in tree-networks and a linear program to compute worst-case performances in general topologies. This approach outperforms the existing approaches both for the stability condition and the performances.

One open question is the choice of the decomposition of the network, the influence of this choice on the stability region and bounds obtained. One way to choose a good decomposition could be to follow the approach of [17] and use neural networks.

Finally, we assumed arbitrary multiplexing, and future work includes adapting the result to specific service policies, such as FIFO, priorities, or generalized processor sharing.

References

1. Amari, A., Mifdaoui, A., Frances, F., Lacan, J., Rambaud, D., Urbain, L.: AeroRing: Avionics Full Duplex Ethernet Ring with High Availability and QoS Manage-

ment. In: ERTS (2016)
2. Amari, A., Mifdaoui, A.: Worst-case timing analysis of ring networks with cyclic dependencies using network calculus. In: RTCSA. pp. 1–10 (2017). https://doi.org/10.1109/RTCSA.2017.8046319
3. Andrews, M.: Instability of FIFO in session-oriented networks. In: Proceedings of SODA'00 (2000)
4. Andrews, M.: Instability of FIFO in the permanent sessions model at arbitrarily small network loads. In: Proceedings of SODA'07 (2007)
5. Bondorf, S., Nikolaus, P., Schmitt, J.B.: Quality and cost of deterministic network calculus – design and evaluation of an accurate and fast analysis. In: ACM SIGMETRICS (2017). https://doi.org/10.1145/3078505.3078594
6. Bondorf, S., Nikolaus, P., Schmitt, J.B.: Quality and cost of deterministic network calculus – design and evaluation of an accurate and fast analysis. Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS) **1**(1), 34 (2017). https://doi.org/10.1145/3084453
7. Bondorf, S., Schmitt, J.B.: Boosting sensor network calculus by thoroughly bounding cross-traffic. In: Proceedings of INFOCOM 2015 (2015). https://doi.org/10.1109/INFOCOM.2015.7218387
8. Bouillard, A., Boyer, M., Corronc, E.L.: Deterministic Network Calculus: From Theory to Practical Implementation. Wiley-ISTE (2018)
9. Bouillard, A., Gaujal, B., Lagrange, S., Thierry, E.: Optimal routing for end-to-end guarantees using network calculus. Performance Evaluation **65**(11-12), 883–906 (2008). https://doi.org/10.1016/j.peva.2008.04.008
10. Bouillard, A., Jouhet, L., Thierry, E.: Tight Performance Bounds in the Worst Case Analysis of Feed Forward Networks. In: INFOCOM'10 (2010). https://doi.org/10.1109/INFCOM.2010.5461912
11. Bouillard, A.: Python package ncbounds (2019), `https://github.com/nokia/NCBounds`
12. Bouillard, A., Nowak, T.: Fast symbolic computation of the worst-case delay in tandem networks and applications. Perform. Eval. **91**, 270–285 (2015). https://doi.org/10.1016/j.peva.2015.06.016
13. Boyer, M., Navet, N., Olive, X., Thierry, E.: The PEGASE project: precise and scalable temporal analysis for aerospace communication systems with network calculus. In: ISOLA'10 (2010). https://doi.org/10.1007/978-3-642-16558-0_13
14. Chang, C.S.: Performance Guarantees in Communication Networks. TNCS, Springer-Verlag (2000)
15. Cruz, R.: A calculus for network delay, part II: Network analysis. IEEE Transactions on Information Theory **37**(1), 132–141 (1991). https://doi.org/10.1109/18.61110
16. Cruz, R.: Quality of service guarantees in virtual circuit switched networks. IEEE Journal on selected areas in communication **13**, 1048–1056 (1995). https://doi.org/10.1109/49.400660
17. Geyer, F., Bondorf, S.: DeepTMA: Predicting effective contention models for network calculus using graph neural networks. In: (INFOCOM) (2019). https://doi.org/10.1109/INFOCOM.2019.8737496
18. Le Boudec, J.Y., Thiran, P.: Network Calculus: A Theory of Deterministic Queuing Systems for the Internet, vol. LNCS 2050. Springer-Verlag (2001). https://doi.org/10.1007/3-540-45318-0, revised version 4, May 10, 2004
19. Le Boudec, J.: A theory of traffic regulators for deterministic networks with application to interleaved regulators. IEEE/ACM Trans. Netw. **26**(6), 2721–2733 (2018). https://doi.org/10.1109/TNET.2018.2875191

20. McManus, J.M., Ross, K.W.: Video-on-demand over ATM: Constant-rate transmission and transport. IEEE J.Sel. A. Commun. **14**(6), 1087–1098 (Sep 1996). https://doi.org/10.1109/49.508280
21. Mohammadpour, E., Stai, E., Mohiuddin, M., Boudec, J.L.: Latency and backlog bounds in time-sensitive networking with credit based shapers and asynchronous traffic shaping. In: 30th International Teletraffic Congress, ITC. pp. 1–6 (2018). https://doi.org/10.1109/ITC30.2018.10053
22. Pellegrini, F.D., Starobinski, D., Karpovsky, M.G., Levitin, L.B.: Scalable cycle-breaking algorithms for gigabit ethernet backbones. In: Proceedings IEEE INFOCOM (2004). https://doi.org/10.1109/INFCOM.2004.1354641
23. Rizzo, G., Boudec, J.Y.L.: Stability and delay bounds in heterogeneous networks of aggregate schedulers. In: Proceedings of INFOCOM'2008 (2008). https://doi.org/10.1109/INFOCOM.2008.208
24. Schmitt, J., Zdarsky, F., Fidler, M.: Delay Bounds under Arbitrary Multiplexing: When Network Calculus Leaves You in the Lurch ... In: INFOCOM'08 (2008). https://doi.org/10.1109/INFOCOM.2008.228
25. Starobinski, D., Karpovsky, M.G., Zakrevski, L.: Application of network calculus to general topologies using turn-prohibition. In: Proceedings IEEE INFOCOM (2002). https://doi.org/10.1109/INFCOM.2002.1019365
26. Tassiulas, L., Georgiadis, L.: Any work-conserving policy stabilizes the ring with spatial re-use. IEEE/ACM Trans. Netw. **4**(2), 205–208 (1996). https://doi.org/10.1109/INFCOM.1994.337631
27. Time-sensitive networking task group, http://www.ieee802.org/1/pages/tsn.html