Stability and performance guarantees in networks with cyclic dependencies

Anne Bouillard

Stability and performance guarantees in networks with cyclic dependencies

Anne Bouillard
Nokia Bell Labs France
Email: Anne.Bouillard@nokia-bell-labs.com

October 5, 2018

Abstract

With the development of real-time networks such as reactive embedded systems, there is a need to compute deterministic performance bounds. This paper focuses on the performance guarantees and stability conditions in networks with cyclic dependencies in the network calculus framework. We first propose an algorithm that computes tight backlog bounds in tree networks for any set of flows crossing a server. Then, we show how this algorithm can be applied to improve bounds from the literature fir any topology, including cyclic networks. In particular, we show that the ring is stable in the network calculus framework.

1 Introduction

With the development of critical embedded systems, it becomes a necessity to compute 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 [15], Video-on-Demand [19]... More recently, it has been very useful for analysis large embedded networks such as AFDX (Avionics Full Duplex) [12].

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 [2]. As a consequence, there is a strong need to design efficient methods for computing precise deterministic bounds.

Recent works ([10, 11, 7]) have focused on computing tight performance bounds in feed-forward networks, but 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 *stopped-time* method. It has first been presented in [14]. A sufficient condition for stability is obtained as the existence of a fix point in an equation derived from the network description.

The theoretical aspects of the stability in deterministic queueing networks have also been studied in the slightly different model named *adversarial queueing network* (see [8] for a precise presentation). An injection rate per server (the rate at which data crossing this server is sent into the network) is given instead of one arrival rate per flow. Stability is stated in function of the topology as a minor-exclusion conditions in [1], in function of a service policy in [5] or of the injection rate in [18].

Fewer works concern the stability in Network calculus. The most classical result is the stability of the ring, which is proved in [23] for work-conserving links and generalized in [17].

Instability results are provided in [4, 3]. In [4], the authors even show that the FIFO policy can be unstable at arbitrary small utilization rates. Some works, such as [21], have focused on finding sufficient condition for the stability in FIFO networks.

Another direction of research has consisted in breaking the cyclic dependencies in order to ensure the stability. Removing arcs could disconnect the network, but forbidding some paths of length 2, as in the *turn-prohibition* method, [22, 20], can ensure both stability and connectivity.

Contributions In this paper, we study the problem of stability in networks with cyclic dependencies in the network calculus formalism, using recent the developments in [11] for tandem networks. Our main contributions are the following:

- we generalize the recent algorithm of [11], that computes exact worst-case delays in a tandem network. We adapt it 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 [11] can be deduced from this new algorithm, while the reverse is not true. As a by-product, we improve the results of [6] about sink-tree networks;
- this new algorithm is used to compute new stability sufficient conditions in networks as mathematical programs. In particular, we demonstrate the stability of the ring in the network calculus framework. A weaker result had already been proved in [17] and [23], but our weaker assumption close a long-standing conjecture.

The rest of the paper is organized as follows: in Section 2, we recall the network calculus basics. Then in Section 3, we present generic mathematical programming methods to compute sufficient condition for the stability of networks. Next in Section 4, we give our algorithm that computes exact worst-case backlog in tree networks. We finally combine these results to compute new stability sufficient conditions in Section 5, and we compare them through numerical experiments in Section 6.

2 Network calculus framework and model

We denote by $\mathbb{N}$ the set of non-negative integers $\{0, 1, \ldots\}$ and for all $n \in \mathbb{N}$, we set $\mathbb{N}_n = \{1, \ldots, n\}$. For $x \in \mathbb{R}$, we set $(x)_+ = \max(x, 0)$. We write $\mathbb{R}_+$ for the set of non-negative reals.

While our model is in line with the standard definitions of networks calculus, we present only the material that is needed in this paper. A more complete presentation of the network calculus framework can be found in the reference books [17, 13].

2.1 Flow and server model

Data flows Flows of data are represented by non-decreasing and left-continuous functions that model the cumulative processes. More precisely, if A represents a flow at a certain point in the network, $A(t)$ is the amount of data of this flow that crossed this point in the time interval $[0, t)$, with the convention $A(0) = 0$. More formally, let

$$\mathcal{F} = \{f : \mathbb{R}_+ \to \mathbb{R} \mid f(0) = 0, f \text{ non-decreasing and left-continuous}\}.$$

A system $\mathcal{S}$ is a non-deterministic relation between input and output flows, where the number of inputs is the same as the number of outputs: $\mathcal{S} \subseteq \mathcal{F}^m \times \mathcal{F}^m$ and there is a one-to-one relation between the inputs and the outputs of the system, such that to each input flow corresponds

one and only one output flow that is causal – no data is created or lost inside the system – meaning that for $((A_i)_{i=1}^m, (B_i)_{i=1}^m) \in \mathcal{S}$, $\forall i \in \mathbb{N}_m$, $A_i \geq B_i$. The vector $((A_i)_{i=1}^m, (B_i)_{i=1}^m)$ is an *(admissible) trajectory* of $\mathcal{S}$ if $((A_i)_{i=1}^m, (B_i)_{i=1}^m) \in \mathcal{S}$. If $m = 1$, i.e., the system has exactly one incoming and one outgoing flow, then we will also refer to it as a *server*.

Arrival curves The notion of arrival curve is quite simple: The amount of data that arrived during an interval of time is a function of the length of this interval. More formally, let $\alpha \in \mathcal{F}$. A flow is constrained by the arrival curve α, or is α-constrained if

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

A typical example of such arrival curve are the pseudo-linear *token-bucket* functions: $\alpha_{b,r} : 0 \mapsto 0;\ t \mapsto b + rt$, if $t > 0$. The burstiness parameter b can be interpreted as the maximal amount of data that can arrive simultaneously and the rate r as a maximal long-term arrival rate.

Service curves The role of a service curve is to constrain the relation between the input of a server and its output. Let A be an cumulative arrival process to a server and B be its cumulative departure process. Several types of service curves have been defined in the literature, and the main types are the simple and strict service curves, which we now define.

We say that β is a *simple service curve* for a server $\mathcal{S}$ if

$$\forall (A, B) \in \mathcal{S}\colon \quad A \geq B \geq A * \beta,$$

with the convolution $A * \beta(t) = \inf_{0 \leq s \leq t} A(s) + \beta(t - s)$.

An interval I is a *backlogged period* for $(A, B) \in \mathcal{F} \times \mathcal{F}$ if $\forall u \in I$, $A(u) > B(u)$. The start of the backlogged period of an instant $t \in \mathbb{R}_+$ is $\text{start}(t) = \sup\{u \leq t \mid A(u) = B(u)\}$. As both A and B are left-continuous, we have $A(\text{start}(t)) = B(\text{start}(t))$, and $(\text{start}(t), t]$ is always a backlogged period.

We say that β is a *strict service curve* for server $\mathcal{S}$ if

$$\begin{aligned} &\forall (A, B) \in \mathcal{S}\colon\ A \geq B \quad \text{and} \\ &\forall \text{ backlogged periods } (s, t]\colon\ B(t) - B(s) \geq \beta(t - s). \end{aligned} \tag{1}$$

We define $\mathcal{S}(\beta)$ as the set of functions (A, B) satisfying Equation (1). The name strict service curves implies a difference to simple service curves. Works on the comparisons between the different types of service curves can be found in [17, 9]. In this article, we will mainly deal with strict service curves, but will make use of the convolution $A * \beta$ used to define simple service curves.

A typical example of a service curve are the *rate-latency* functions: $\beta_{R,T} : t \mapsto R \cdot (t - T)_+$, where T is the latency until the server has to become active and R is its minimal service rate after this latency.

Note that a server $\mathcal{S}$ may not be deterministic, as the function β only corresponds to a guarantee on the service offered. Among this non-determinism, we will focus on two modes of operation:

- **Exact service mode:** During a backlogged period $(s, t]$, the service is *exact* if for all $u \in (s, t]$ we have $B(u) = A(\text{start}(t)) + \beta(u - \text{start}(t))$. (In this case, $\text{start}(u) = \text{start}(t)$.)
- **Infinite service mode:** During an interval of time $(s, t]$, the service is *infinite* if $\forall u \in (s, t]\colon B(u) = A(u)$, i.e., the server serves all data instantaneously.

For a system $\mathcal{S} \in \mathcal{F}^m \times \mathcal{F}^m$ with m inputs and outputs, we say that $\mathcal{S}$ offers a strict service curve β if the aggregate system is, that is, if

$$\forall\, ((A_i)_{i\in\mathbb{N}_m}, (B_i)_{i\in\mathbb{N}_m}) \in \mathcal{S},\ \left(\sum_{i\in\mathbb{N}_m} A_i, \sum_{i\in\mathbb{N}_m} B_i\right) \in \mathcal{S}(\beta).$$

We assume no knowledge about the service policy in this system (except that it is FIFO per flow).

2.2 Performance guarantees

Backlog In this article, we focus on the worst-case backlog of a flow or a set of flow at a given server of a network. Let $(A, B) \in \mathcal{F}^2$ be an admissible trajectory of a server. The backlog of the server at time t is $b(t) = A(t) - B(t)\}$. The worst-case backlog is then $b_{\max} = \sup_{t\geq 0} b(t)$. Graphically this is the maximal vertical distance between A and B.

We denote $\ell(t) = t - \text{start}(t)$, the length of the backlogged period upt to t and $\ell_{\max} = \max_t \ell(t)$, the maximum length of a backlogged period.

We denote $b_{\max}(\alpha, \beta)$ (resp. $\ell_{\max}(\alpha, \beta)$) the maximum backlog (resp. the maximum length of a backlogged period) that can be obtained for a flow that is α-constrained crossing $\mathcal{S}(\beta)$. For example, we have

- $b_{\max}(\gamma_{b,r}, \beta_{R,T}) = b + rT$ if $r \leq R$, $= +\infty$ otherwise;
- $\ell_{\max}(\gamma_{b,r}, \beta_{R,T}) = \frac{b+RT}{R-r})$ if $r < R$, $= +\infty$ otherwise.

For a system with m inputs and outputs, it is also possible to compute the maximal backlog for a set of flows crossing this server. If $I \subset \mathbb{N}_m$, the backlog of flows in I at time t is $b_I(t) = \sum_{i\in I} F_i^{(in)}(t) - \sum_{i\in I} F_i^{(out)}(t)$. If the system offers a strict service curve β and flow i is α_i-constrained, then

$$b_I(t) \leq b_{I,\max} = (\sum_{i\in I} \alpha_i) \oslash (\beta - \sum_{j\notin I} \alpha_j)_+(0),$$

where $f \oslash g(t) = \sup_{u\geq 0} f(t+u) - g(u)$ is the (min,plus)-deconvolution. In the case of leaky-bucket arrival curves and rate latency service curve,

$$b_{I,\max} = b_I + \frac{r_I}{R - r_{\overline{I}}}(b_{\overline{I}} + r_{\overline{I}}T) + r_I T, \tag{2}$$

with $r_I = \sum_{i\in I} r_i$, and similarly for $r_{\overline{I}}$, b_I and $b_{\overline{I}}$.

Stability We will also be interested in the stability of a network. Let us first define the stability for server:

Definition 1 (Server stability)**.** *Consider a server offering a strict service curve β and a flow crossing it, with arrival curve α.*

- *This server is said* unstable *if its worst-case backlog is unbounded;*
- *This server is said* critical *if its worst-case backlog is bounded, but the lengths of its backlogged periods are not bounded;*
- *This server is said* stable *if the length of its backlogged periods is bounded.*

If the service curve is rate-latency $\beta_{R,T}$ and the arrival curve token-bucket $\gamma_{b,r}$, then a server is unstable if $R < r$, critical is $R = r$ and stable if $R > r$.

Note that this definition only involves r and R. The stability is insensitive to b and T, that only influence the size of the backlog and backlogged period.

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 rate-latency strict service curve $\beta^{(j)} = \beta_{R_j,T_j}$;
- each flow crosses the network along a path $\pi = \langle \pi_i(1), \ldots, \pi_i(\ell_i) \rangle$, where $\ell_i \geq 1$ is the length of the path. Each flow is constrained by the arrival curve $\alpha_i = \gamma_{b_i,r_i}$.

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

We denote by $\mathcal{N}$ this network. Its induced graph is the directed graph whose vertices are the servers and the set of arcs is

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

We assume, without loss of generality, as we will focus on the performances in one server, that the network is connected has a unique final strictly connected component. Moreover, we assume that for all $j \in \mathbb{N}_{n-1}$, there exists $j' > j$ such that $(j, j') \in \mathbb{A}$ (up to renumbering the servers, this is also without loss of generality).

Classes of networks:

- if $\mathbb{A} = \{(j, j+1) \mid j \in \mathbb{N}_{n-1}\}$, then the network is called an *tandem networks*;
- if the output degree of each vertex except node n is 1, then the network is called a *tree network*;
- if the graph network has no cycle, the network is called *feed-forward*;
- otherwise, it has *cyclic dependencies*.

Network stability

Definition 2 (Local stability)**.** *Consider a network $\mathcal{N}$. It is said locally stable if all its servers are stable using the initial arrival curves: $\forall j \in \mathbb{N}_n$,*

$$\ell_{\max}(\sum_{i \ni j} \alpha_i, \beta^{(j)}) < \infty.$$

Definition 3 (Global stability)**.** *A network is* globally stable *if for all its servers, the length of the maximal backlogged period is bounded.*

We call the *linear model* when arrival curves are leaky-bucket and the service curve rate-latency. Our aim is to give sufficient properties for the global stability for networks with cyclic dependencies (the underlying graph has cycles).

Lemma 1. *If a network is globally stable, then it is locally stable.*

Proof. We prove this by contra-position: suppose that the network is not locally stable. Then, there exists a server j that is not stable considering the original arrival process. Consider the following trajectory: every server acts as an infinite server except server j. Then the arrival processes into server j are exactly those that are injected into the network, and server j is not stable: the length of its backlogged period cannot be bounded. But they cannot either in this behavior of the network. So the network is not globally stable. □

3 Sufficient conditions of network stability

When the network is feed-forward, it is possible to compute the performance of the network by

1. applying Equation (2) at every server and for $I = \{i\}$ for each flow f_i crossing that server and
2. propagating the constraints in the topological order of the servers.

This is not possible when there are cyclic dependencies, because of the inter-dependencies between the backlogs computed when applying Equation (2).

The fix-point method is a generic method to compute performance guarantees in networks with cyclic dependencies. The main idea is to compute, for each server and each flow crossing it, an output arrival curve that depends on the input arrival curves at that server. A system of equations is then obtained and the solution, if it exists, gives output arrival curves for each flow after each server it crosses. This approach has been described for leaky bucket arrival curves and rate-latency service curves in [17]. The method for proving this approach is the *stopped-time* method.

In this section, we generalize the stopped-time method described in [17] in four directions:

- it can be applied to any arrival and service curve;
- it can be applied to any feed-forward decomposition of the network rather than a for each server individually;
- it can be applied to any group of flows rather than considering each flow individually;
- it can handle different combinations of the two cases above.

Instead of computing a fix-point, we rather present our results as the solution of an optimization problem, that enables us to include the computation of the network performance (worst-case backlog for example) in problem. In the linear model, we obtain a linear program.

With the three first points of generalization, it is also possible to solve a fix-point problem in order to obtain stability sufficient conditions. But the fourth point has no natural formalization into a fix-point problem.

We first describe the transformation of the network and flow grouping before and computing performance bounds as an optimization problem.

3.1 Network transformation

3.1.1 Feed-forward decomposition

Let $\mathcal{N}$ be a network and $G_{\mathcal{N}}$ be its induced graph. This graph can be transformed into an acyclic graph by removing a set of arc $\mathbb{A}^{\mathrm{r}} \subseteq \mathbb{A}$.

Example 1. *The toy network of Fig. 1 can be transformed into an acyclic network by removing arc* $(4, 2)$*. In the next section, we will see that [erformance bounds can be efficiently computed in tree topologies. Such a decomposition can be obtained with* $\mathbb{A}^{\mathrm{r}} = \{(4, 2), (2, 1)\}$*. With* $\mathbb{A}^{\mathrm{r}} = \mathbb{A}$*, all arcs are removed, and we obtain a graph with isolated nodes only.*

Note that this transformation is not unique, and finding the minimum set of edges to remove is a NP-complete problem (it is the *Minimum feed-back arc set* problem in [16]). The most classical solution in the literature is to remove every arc – each server is analyzed in isolation –

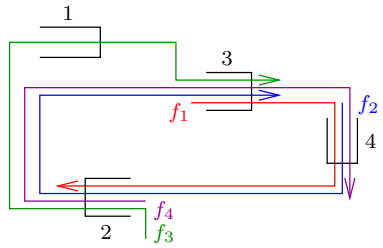

Figure 1: Toy network of Example 1.

but it might be a better choice to remove fewer arcs, and obtain a tree for example, we will see later that these topologies can be easily analyzed.

We now modify the flows in accordance to the arcs that have been removed: each flow f_i is split into several flows $f_{i,1}, f_{i,2}, ..., f_{i,m_i}$ of respective paths in $(\mathbb{N}_n, \mathbb{A}-\mathbb{A}^{\mathrm{r}})$, $\pi_{i,1} = \langle \pi_i(1), ..., \pi_i(k_1) \rangle$, $\pi_{i,2} = \langle \pi_i(k_1+1), ..., \pi_i(k_2) \rangle$,..., $\pi_{i,m_i} = \langle \pi_i(k_{m_i}+1), ..., \pi_i(\ell_i) \rangle$, where $(\pi_i(k_j+1), \pi_i(k_j)) \in \mathbb{A}^r$. We denote $\ell_{i,k}$ the length of $\pi_{i,k}$, and call $\mathcal{N}^{\mathrm{FF}}$ the feed-forward network that is obtained.

Example 2. *In the toy example of Fig. 1, if* $\mathbb{A}^{\mathrm{r}} = \{(4,2),(2,1)\}$*, flow* f_1 *is split into* $f_{1,1}$ *and* $f_{1,2}$ *with respective paths* $\langle 3,4 \rangle$ *and* $\langle 2 \rangle$*, flow* f_2 *is split into* $f_{2,1}$ *and* $f_{2,2}$*, with respective paths* $\langle 4 \rangle$ *and* $\langle 2,3 \rangle$ *and flow* f_3 *is split into* $f_{3,1}$ *and* $f_{3,2}$*, with respective paths* $\langle 2 \rangle$ *and* $\langle 1,3 \rangle$*. Flow* f_4 *remains unchanged* ($f_{4,1} = f_4$)*. The result of this decomposition is depicted Fig. 2.*

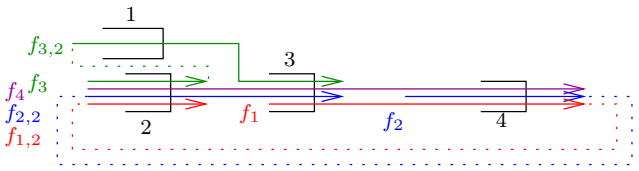

Figure 2: Feed-forward transformation of the toy network.

3.1.2 Flow grouping

The second step is to group flows. Instead of dealing with flows $f_{i,k}$ individually, we use a partition. Let $S = \{(i,k) \mid i \in \mathbb{N}_m,\ k \in \mathbb{N}_{m_i}\}$ be the set of flows that have been created with the feed-forward transformation and $S_1, \ldots, S_L \subseteq S$ with $S_1 \sqcup S_2 \sqcup \cdots \sqcup S_L = S$ (the symbol $\sqcup$ denotes the disjoint union).

Our goal is to compute $(\alpha^\ell)_{\ell \in \mathbb{N}_L}$ where α^ℓ is an arrival curves of the aggregation of the flows $(f_s)_{s \in S_\ell}$ in S_ℓ. We denote f^ℓ the aggregated flow.

This formulation might at first seem quite strange, but it takes several interesting cases into account.

Example 3. *Consider again the toy example with the decomposition of Fig. 2. There is a priori no use to group flows* $f_{i,1}$*, as their arrival curve is already known. For the other flows, there are two natural groupings. The first is to group individually: for all* ℓ*,* S_ℓ *is a singleton. The second one is to group according to the removed arcs, and have the grouping* $\{(1,2),(2,2)\}$ *and* $\{(3,2)\}$*. Indeed, the arrival curve of* $f_{1,2} + f_{2,2}$ *can be less than the sum of the arrival curves of each individual flow, so hopefully, better performances can be computed with this decomposition.*

3.2 Stability and performances as an optimization problem

If $\mathcal{N}$ is stable, then there exists an arrival curve α^ℓ for the aggregated flow f^ℓ.

We make the following assumptions:

($\mathbf{A}_1$) For all $\ell \in \mathbb{N}_L$, there exists a non-decreasing function $H_\ell : \mathcal{F}^L \to \mathcal{F}$ that computes an arrival curve in $\mathcal{N}^{\mathrm{FF}}$ for the aggregation of flows $(f_{i,k-1})_{(i,k)\in S_\ell}$ at the end of their respective path $\pi_{i,k-1}$ in function of $\alpha^1, \ldots, \alpha^L$, the respective arrival curves of all aggregated flows f^ℓ.

($\mathbf{A}_2$) There exists a non-decreasing function $G : \mathcal{F}^L \to \mathbb{R}_+$ that computes a performance P in $\mathcal{N}$ as a function of $(\alpha^\ell)_{\ell=1}^L$, arrival curves for the aggregate flows f^ℓ.

($\mathbf{A}_3$) $(H_\ell)_{\ell=1}^L$ and G implicitly depend on the arrival curves $(\alpha_i)_{i=1}^m$ and on the service curves $(\beta_j)_{j=1}^n$. These functions are assumed to be non-decreasing with α_i, $i \in \mathbb{N}_m$ and non-increasing with β_j, $j \in \mathbb{N}_n$.

Assumptions ($\mathbf{A}_1$) and ($\mathbf{A}_2$) are ensured when $\mathcal{N}^{\mathrm{FF}}$ is feed-forward. It corresponds to choosing a method for computing performances in feed-forward networks. Assumption ($\mathbf{A}_3$) and the fact that H_ℓ and G are non decreasing are made without loss of generality since with greater arrival curves and smaller service curves induce more admissible trajectories, hence larger worst-case performances.

Since the minimum of two arrival curves for a flow is also an arrival curve for that flow, when $\mathcal{N}$ is stable, we can consider that α^ℓ is the minimum arrival curves for f^ℓ. Then it holds $\forall \ell \in \mathbb{N}_L$, $\alpha^\ell \leq H_\ell(\boldsymbol{\alpha})$, where $(\alpha^\ell)_{\ell=1}^L = \boldsymbol{\alpha}$. We can write this latter inequality as a vector expression:

$$\boldsymbol{\alpha} \leq H(\boldsymbol{\alpha}), \tag{3}$$

where $H = (H_\ell)_{\ell=1}^L$. The next theorem shows the reverse: if the solutions of Equation (3) are bounded, then the system is stable.

Theorem 1. *Set $\mathcal{C} = \{\boldsymbol{\alpha} \mid \boldsymbol{\alpha} \leq H(\boldsymbol{\alpha})\}$ be the set of solutions of Equation (3), and $\boldsymbol{\alpha}_0 = \sup\{\boldsymbol{\alpha} \mid \boldsymbol{\alpha} \in \mathcal{C}\}$. If $\boldsymbol{\alpha}_0$ is finite, then $\mathcal{N}$ is globally stable and for all $\ell \in \mathbb{N}_L$, α_0^ℓ is an arrival curve for the aggregation of flows f_i at the input of server $\pi_{i,k}(1)$ for $(i,k) \in S_\ell$.*

The proof of this theorems follows exactly the same lines as the stopped-time method described in [17] or [13].

Proof. First, $\boldsymbol{\alpha}_0$ exists, as $\mathcal{C}$ contains a maximum element: if $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$ are two elements in $\mathcal{C}$, then for all $\ell \in \mathbb{N}_L$, $\alpha_1^\ell \leq H_\ell(\boldsymbol{\alpha}_1) \leq H_\ell(\boldsymbol{\alpha}_1 \vee \boldsymbol{\alpha}_2)$ and similarly, $\alpha_2^\ell \leq H_\ell(\boldsymbol{\alpha}_1 \vee \boldsymbol{\alpha}_2)$, so $\alpha_1^\ell \vee \alpha_2^\ell \leq H_\ell(\boldsymbol{\alpha}_1 \vee \boldsymbol{\alpha}_2)$, and $\boldsymbol{\alpha}_1 \vee \boldsymbol{\alpha}_2 \in \mathcal{C}$.

We use the stopped-time method. Consider that the arrivals to the network stop at time $\tau > 0$: for each flow f_i, an arrival curve is then $\alpha_i^\tau : t \mapsto \alpha_i(t \wedge \tau)$. The total amount of data for each flow f_i is also bounded by $\alpha_i(\tau)$, so the network is globally stable.

Let $\boldsymbol{\alpha}^\tau = (\alpha^{\tau,\ell})_{\ell\in\mathbb{N}_L}$ be the family of the *minimal arrival curve* of the aggregated flows f_i at server $\pi_{i,k}(1)$ for $(i,k) \in S_\ell$ then

$$\boldsymbol{\alpha}^\tau \leq H^\tau(\boldsymbol{\alpha}^\tau) \leq H(\boldsymbol{\alpha}^\tau),$$

where H^τ is obtained the same way as H, but replacing α_i by α_i^τ. The first inequality comes from the stability of $\mathcal{N}$ for the stopped process, and the second from Assumption ($\mathbf{A}_3$).

For all $\tau > 0$, $\boldsymbol{\alpha}^\tau \in \mathcal{C}$, so $\boldsymbol{\alpha}_0 \geq \boldsymbol{\alpha}^\tau$, and $\boldsymbol{\alpha}_0$ is a family of arrival curves that is valid for all $\tau > 0$. Then it is valid for the whole unstopped process, and the system is stable if $\boldsymbol{\alpha}_0$ is finite. □

3.2.1 One-stage optimization problem

We are now ready to give a mathematical programming problem that computes worst-case performances upper bounds for arbitrary networks:

$$\boxed{\text{Maximize } G(\boldsymbol{\alpha}) \text{ such that } \boldsymbol{\alpha} \leq H(\boldsymbol{\alpha}).} \tag{4}$$

Theorem 2. *Under Assumptions ($\mathbf{A}_1$), ($\mathbf{A}_2$) and ($\mathbf{A}_3$), the solution of the optimization problem of Equation* (4) *is an upper bound of the performance P of the network.*

Proof. As H_ℓ and G are non-decreasing, $\boldsymbol{\alpha}^0$ maximizes G among all the elements such that $\boldsymbol{\alpha} \leq H(\boldsymbol{\alpha})$. □

This formulation is in fact equivalent to the *fix-point method* that can be found in the literature, and that can be deduced from Theorem 1. Indeed, the proof of that theorem ensures the existence of a greatest fix-point. As $\boldsymbol{\alpha}_0$ is that fix-point, then the performances can be directly computed as $G(\boldsymbol{\alpha}_0)$.

This result is often used when there exists a unique fix-point to the equation $\boldsymbol{\alpha} = H(\boldsymbol{\alpha})$, which is also the largest solution of $\boldsymbol{\alpha} \leq H(\boldsymbol{\alpha})$. Our formulation enables to apply the fix-point method in cases the uniqueness is not ensured.

3.2.2 Two-stage optimization problem

The formulation as an optimization problem can be generalized, by making advantage of two decompositions. For example, one could have a feed-forward transformation of the network so that $\mathcal{N}^{\text{FF}}$ is a tree. Following example 3, there are two natural ways to group flows. First considering singletons only, which defines functions G and H_1; second grouping flows according to the arc that has been removed, which define functions H_2.

So the following mathematical program is obtained:

$$\boxed{\begin{array}{ll} \text{Maximize } G(\boldsymbol{\alpha}) & \\ \quad \text{such that} & \boldsymbol{\alpha} \leq \boldsymbol{\alpha}_1,\ \boldsymbol{\alpha} \leq \boldsymbol{\alpha}_2 \\ & \alpha_1^s \leq \alpha_2^\ell \leq \sum_{u \in S_\ell} \alpha_1^u,\ \forall s \in S^\ell \\ & \boldsymbol{\alpha}_1 \leq H_1(\boldsymbol{\alpha}_1),\ \boldsymbol{\alpha}_2 \leq H_2(\boldsymbol{\alpha}_2) \end{array}} \tag{5}$$

Theorem 3. *Under Assumptions ($\mathbf{A}_1$), ($\mathbf{A}_2$) and ($\mathbf{A}_3$), the solution of the optimization problem of Equation* (5) *is an upper bound of the performance P of the network.*

Proof. Let $\boldsymbol{\alpha}_1$ be the vector of the smallest arrival curves for the individual flows, and $\boldsymbol{\alpha}_2$ for the aggregated flows. If $s \in S_\ell$, then $\alpha_s \leq \alpha^\ell$, as flow f_s is part of the aggregated flow f^ℓ, and α^ℓ is less than the sum of the arrival curves of all the aggregated flows. Hence, $\alpha_1^s \leq \alpha_2^\ell \leq \sum_{u \in S_\ell} \alpha_1^u$ is satisfied. □

The problem of Eq (5) has no natural equivalent as a fix-point equation, and can be directly generalized for more than two stage, and different decompositions. We will see that this optimization problem is slightly improved in the linear model.

4 Worst-case backlog in tree networks

In this section, we focus on tree networks and give an algorithm to compute exact worst-case backlog in the linear model. The algorithm is a generalization of the one given in [11] 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, so we skip the detailed proof here. The complete proof is in Appendix A.

Let us first give some additional notations used in the algorithm to describe a tree network. First, its induced graph is a tree directed to vertex n, whose output degree 0. Each other vertex j has output degree 1. We denote by $j^\bullet$ its successor and assume that $j < j^\bullet$ and set $n^\bullet = n+1$ by convention. The set of predecessor of a vertex is ${}^\bullet j = \{k \mid k^\bullet = j\}$. There exists at most one path between two vertices j and k, denoted $j \rightsquigarrow k$. Finally, if there exists a path from j to k, ${}^{\bullet j}k$ is the predecessor of k of this path.

Suppose that we are interested in computing the worst-case backlog at server n for some flows crossing it. We denote by $I \subseteq \mathbb{N}_m$ those flows of interest.

- $r_j^k = \sum_{i \in \mathrm{Fl}(j) \setminus I, \pi_i(\ell_i)=k} r_i$ 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 interests 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 $\varphi_i \leftarrow \xi_{\pi_i(1)}^{\pi_i(\ell_i)}$;

Theorem 4. *Consider a tree network with n servers offering 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*

$$B = \sum_{j=1}^{n} \rho_j T_j + \sum_{i=1}^{m} \varphi_i b_i, \tag{6}$$

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 $O(n^2)$.

Sketch of the proof. The proof of the theorem is based on the construction of an admissible trajectory (*i.e.* cumulative functions for each flow, at the input/output of each server in the path of this flow, that respect the input and output constraints given by the arrival and service curves) whose backlog at server n is maximal for the flows in I (we call it a *worst-case trajectory*).

Similar to the proof in [11], the proof is in two steps. First, we show that there exists a worst-case trajectory that satisfy some properties. The second step is to construct a worst-case trajectory among the trajectories having those properties.

Properties of a worst-case trajectory: suppose that the worst case backlog is obtained at time $t_{n+1} = t_{n\bullet}$. There exists a worst-case trajectory that satisfy the following properties.

1. The service policy is SDF (shortest-to-destination-first).
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.
3. The arrival function of flow f_i entering the system at server j is maximal from t_j, the start of the backlogged period of server j for all $t > t_j$ and 0 otherwise.
4. Data from the flows of interest in I crossing server j are instantaneously served at time $t_{j\bullet}$ and are all in server n at time t_{n+1}.

These properties are straightforward generalizations from [11] to trees.

Worst-case trajectory with the properties: Once the set of trajectories has been restricted to the one satisfying the four properties above, the only optimization remaining is choosing the dates t_j. Indeed, if dates t_j are fixed, the four properties above exactly determines the trajectory. Intuitively, the larger the backlog transmitted to the next server, the larger the backlog at server n. The maximization of the transmitted backlogs is done by a backward induction, from the root of the tree (server n) to the leaves that is detailed in Appendix A. This optimization is then translated into Algorithm 1. □

Worst-case delay The worst-case delay of a flow can be deduce 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 $I = \{1\}$.

Proof. Suppose the flow of interest is f_1, with starting at server $\pi_1(1) = j$ and ending at server $\pi_1(\ell_1) = n$. We are interested in computing the worst-case delay of this flow. From [11], the worst case delay is obtained for the bit of data b_1. We can then do the following modification: assume there are $m + 1$ flows. Flows f_2 to f_m remain unchanged, flow f_1 has arrival curve

$t \mapsto b_1$ and flow f_0 has arrival curve $t \mapsto r_1 t$. The flow on interest is now flow f_0. The worst-case backlog is obtained at time t_{n+1}, and is $r_1(t_{n+1} - t_j)$. It is maximal when $t_{n+1} - t_j$ is, and then is the worst-case delay for the first bit is data of flow f_0, which is equivalent to bit of data b_1 of the original network.

Let B be the worst-case backlog obtained with Algorithm 1 with the original network. With the modified network, the backlogs becomes $B' = B - b_1 + \xi_j^n b_1$, as the new flow f_1 is not of interest (in the transformation, φ_1 changes from 1 to ξ_j^n), then the worst-case delay is B'/r_1, which corresponds to the desired result. □

Application to sink-trees 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 5-13) 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, the algorithm can be performed in $O(n)$. This type of topology has been studied in the context of *Sensor Network Calculus*. In [6], the authors give a close-form formula for the maximum backlog at the root and the end-to-end delay of a flow of interest.

Concerning the maximum backlog, this corresponds in our algorithm to the case where every flow is a flow of interest, so $\phi_i = 1$ for each flow i and $\rho_j = r_j^*$. It is easy to see that the formula is the same as in [6, Theorem 14].

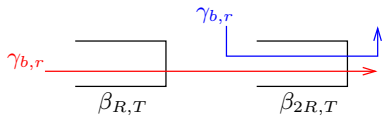

Figure 3: Example of sink-tree.

Concerning the end-to-end delay of a flow, we show on a simple example that our approach leads to tighter delays. Consider the toy example with two servers in tandem described in Figure 3. The worst-case delay bound from [6, Theorems 18 and 15] is

$$D_1 = 2T + \frac{2b + rT}{R}.$$

With our algorithm, we compute $\xi_2^2 = \frac{r}{2R-r}$ and $\xi_1^2 = \frac{r}{R}$, so the worst-case delay is, from Corollary 1,

$$D_2 = 2T + \frac{b}{R} + \frac{b + rT}{2R - r}.$$

As $R > r$, $D_2 < D_1$. This is quite intuitive, as the cross-traffic arrives at server 2, and then is served at rate $2R$.

Arrival curve for the departure processes

Corollary 2. *With the same notations as in Theorem 4, the arrival curve of the departure functions from server n for flows in I is $\gamma_{B,\sum_{i\in I} r_i}$.*

This is a direct consequence of Theorem 4 and of Lemma 2.

Lemma 2. *Consider a system and flows crossing that system being globally constrained by the arrival curve $\gamma_{b,r}$. If the maximal backlog of these flows in this system is less than B, then the arrival curve for the departure process of these flows is constrained by $\gamma_{B,r}$.*

Proof. Let $F^{(in)}$ be the sum of the arrival processes of the flows of interest and $F^{(out)}$ the sum of departure processes. Fix $s < t$, and transform $F^{(in)}$ from time s: for each flow of interest, the arrival process becomes maximal: a burst arrives at time s, and then data arrival at rate r. Call F' this process. We have $F'(s) - F^{(out)}(s) \leq B$, by hypothesis, and $F^{(out)}(t) \leq F^{(in)}(t) \leq F'(s) + r(t-s)$. So $F^{(out)}(t) - F^{(out)}(s) \leq F'(s) + r(t-s) - F^{(out)}(s) \leq B + r(t-s)$. □

5 Stability and performance bounds in cyclic networks

In this section, we combine the results of the two previous sections. We first restrict to the linear model, and in the last paragraph, we show how those results could be extended to more general cases.

5.1 One-stage optimization problem

We first investigate the one-stage optimization problem of Equation (4). Given a network, several transformations are possible, and we give here three of them. Due to the linear model, the optimization problem boils down to a linear program: there exist a non-negative matrix $M \in \mathbb{R}^{L,L}$, a non-negative column-vector $N \in \mathbb{R}^L$, a non-negative line-vector $Q \in \mathbb{R}^L$ and a non-negative constant C such that performance P can be compute as

$$\boxed{\text{Maximize } Q\mathbf{b} + C \text{ such that } \mathbf{b} \leq M\mathbf{b} + N.} \tag{7}$$

A stability condition is given by the following theorem:

Theorem 5. *If the spectral radius of M is strictly less than 1, then $\mathcal{N}$ is stable.*

As a consequence, depending on the decomposition we will obtain different stability conditions. In the following, we only explicit the construction of M and N. Vector Q and constant C can be computed by similar methods.

5.1.1 Server decomposition

In the literature, the most usual decomposition is into elementary servers ($\mathbb{A}^{\mathrm{r}} = \mathbb{A}$) and elementary flows (no grouping). With our notation, $S = \{(i,k) \mid i \in \mathbb{N}_m, 1 \leq k \leq \ell_i\}$ and $\pi_{i,k} = \langle \pi_i(k) \rangle$, and $L = |S| = \sum_{i \in \mathbb{N}_n} \ell_i$. The decomposition of S is into singletons, and if $S^\ell = \{(i,k)\}$, we simply denote by $\alpha_{i,k}$ the smallest arrival curve of $f_{i,k}$.

From classical results (see [17, Sec. 6.3.2] for example), for all $k < \ell_i$,

$$\alpha_{i,k+1} \leq \alpha_{i,k} \oslash (\beta_j - \sum_{s \in S_j \setminus \{(i,k)\}} \alpha_s)_+, \tag{8}$$

and $\alpha_{i,1} = \alpha_i$, which gives for leaky-bucket and rate-latency curves, and under stability assumption, that $\alpha_{i,k} = \gamma_{b_{i,k}, r_i}$ and from Equation (8),

$$b_{i,k+1} \leq b_{i,k} + \frac{r_i}{R_j - \sum_{p \in \mathrm{Fl}(j) \setminus \{i\}} r_p} \Big(\sum_{s \in S_j \setminus \{(i,k)\}} b_s + R_j T_j \Big).$$

Parameters T_j, R_j and r_i are fixed and b_s are variables, this equation gives the coefficients of M and N, that we denote M_{SD} and N_{SD} in the following.

5.1.2 Tree decomposition

In this paragraph, we use the decomposition into a tree instead of decomposing the network into elementary servers.

Suppose that arcs $\mathbb{A}^{\mathrm{r}}$ have been removed such that the remaining network is a tree networks. In this case, as a tree has exactly $n-1$ arcs, so $L = |S| \leq \sum_{i\in\mathbb{N}_n} \ell_i - n + 1$.

Consider $a = (j_1, j_2) \in \mathbb{A}^{\mathrm{r}}$ and $f_{i,k}$ a flow such that $j_1 = \pi_{i,k}(\ell_{i,k})$ and $j_2 = \pi_{i,k+1}(1)$. An arrival curve for flow $f_{i,k+1}$ can be computed from the others: from Lemma 2, an arrival curve for flow $f_{i,k+1}$ is $\gamma_{b_{i,k+1},r_i}$ where $b_{i,k+1}$ is the maximum backlog for flow i at server j_1 computed with Algorithm 1. As $b_{i,k+1}$ is linear in the bursts of the other flows, there exists $(\varphi_s^{i,k+1})_{s\in S}$ and $(\rho_j^{i,k+1})_{j=1}^n$ such that

$$b_{i,k+1} \leq \sum_{s\in S} \varphi_s^{i,k+1} b_s + \sum_{\{j|j\rightsquigarrow j_1\}} \rho_j^{i,k+1} T_j,$$

where the exponent $i, k+1$ emphasizes the fact that the backlog computed is the burst parameter of $\alpha_{i,k+1}$.

As a consequence, with M_{TD} and N_{TD} playing the role of M and N above, we have $(M_{\mathrm{TD}})_{s,s'} = \varphi_{s'}^s$ and $(N_{\mathrm{TD}})_s = \sum_{\{j|j\rightsquigarrow j_1\}} \rho_j^s T_j$.

Example 4. *Consider the tree decomposition of Figure 2 is obtained.*

To find the other equations, we apply Algorithm 1 and with the notations above,

$$\begin{cases} b_{1,2} = b_{1,1} + \varphi_{2,1}^{1,2} b_{2,1} + \varphi_{1,2}^{1,2} b_{1,2} + Cst_1 \\ b_{2,2} = b_{2,1} + \varphi_{1,1}^{2,2} b_{1,1} + \varphi_{1,2}^{1,2} b_{1,2} + Cst_2 \\ b_{3,2} = b_{3,2} + \varphi_{1,2}^{3,2}(b_{1,2} + b_{2,2} + b_{4,1}) + \rho_2^{3,2} T_2. \end{cases} \tag{9}$$

Note that the expression of $b_{3,2}$ only depends on the behavior of server 2. Indeed, Algorithm 1 only explores a server and its descendants, but server 2 has none. Also, note that from Algorithm 1, two flows following the same path have the same linearity coefficient: $\varphi_{1,2}^{3,2} = \varphi_{2,2}^{3,2} = \varphi_{4,1}^{3,2}$.

As the backlog bounds computed with Algorithm 1 are tight, the stability condition with matrix M_{TD} is better than that with matrix M_{SD}.

5.1.3 Arc grouping

Despite the fact that Algorithm 1 computes the worst-case backlog bound for each flow, M_{TD} having spectral radius less than one is only a sufficient condition for the network stability. Indeed, the linear system is obtained by running Algorithm 1 independently $|S|$ times, but worst-case bounds for flows s and s' ending at the same server do not happen at the same time. Indeed, consider two flows with respective arrival curve γ_{b_1,r_1} and γ_{b_2,r_2} crossing a server offering a strict service curve $\beta_{R,T}$. Then the worst-case delay for flow 1 is $B_1 = b_1 + \frac{r_1}{R-r_2}(b_2 + RT)$, for flow 2 is $B_2 = b_2 + \frac{r_2}{R-r_1}(b_1 + RT)$ and the worst-case backlog in the server is $B = b_1 + b_2 + (r_1 + r_2)T$. Obviously, $B < B_1 + B_2$.

In this paragraph, our strategy is to group flows according to the removed arcs.

Suppose that the network is stable and denote by B_a the worst-case backlog at arc $a = (j_1, j_2) \in \mathbb{A}^{\mathrm{r}}$, that is the maximal backlog at server j_1 of flows having $\langle j_1, j_2\rangle$ as a sub-path. We denote $S_a = \{(i,k) \in S \mid \pi_{i,k}(\ell_{i,k}) = j_1 \text{ and } \pi_{i,k+1}(1) = j_2\}$ and $S'_a = \{(i,k+1) \in S \mid \pi_{i,k}(\ell_{i,k}) = j_1 \text{ and } \pi_{i,k+1}(1) = j_2\}$.

With Algorithm 1, one can compute an upper of bound B_a for each $a \in \mathbb{A}^{\mathrm{r}}$: $\exists \varphi_s^a$ and ρ_j^a such that

$$B_a \leq \sum_{s\in S} \varphi_s^a b_s + \sum_{\{j|j\rightsquigarrow j_1\}} \rho_j^a T_j.$$

The next step is to refine this equation so that $(B_a)_{a\in\mathbb{A}^{\mathrm{r}}}$ appear in the right-hand term instead of $(b_s)_{s\in S}$. We know from the proof of Theorem 4 that the worst-case backlog is maximized when the cross-traffic is maximal. Consider arc $a'=(j'_1,j'_2)\in\mathbb{A}^{\mathrm{r}}$. For all $s\in S'_{a'}$, the arrivals of f_s will all be maximized from time $t_{j'_2}$. At this time, the backlog in server j'_1 is at most $B_{a'}$ and the backlog of each flow transmitted to server j'_2 is x_s with $\sum_{s\in S'_{a'}} x_s \le B_{a'}$. From time $t_{j'_2}$ on, data of flow f_s necessarily arrives at rate r_s: if it could arrive faster, the backlog would not have been maximized.

As a consequence, for all $a'=(j'_1,j'_2)\in\mathbb{A}^{\mathrm{r}}$, if $B_{a'}$ is the worst-case backlog at server j'_s, for flows f_s, $s\in S'_{a'}$, there exists $(x_s)_{s\in S'_a}$ such that $\sum_{s\in S'_{a'}} x_s \le B_{a'}$ and

$$\begin{aligned}
B_a &\le \sum_{s\in S}\varphi^a_s x_s + \sum_{\{j|j\rightsquigarrow j_1\}}\rho^a_j T_j\\
&\le \sum_{a'\in\mathbb{A}'}\Big[\Big(\max_{s\in S'_{a'}}\varphi^a_s\Big)\Big(\sum_{s\in S'_a}x_s\Big)\Big] + \sum_{i=1}^{m}\varphi^a_{(i,1)}b_i + \sum_{\{j|j\rightsquigarrow j_1\}}\rho^a_j T_j\\
&\le \sum_{a'\in\mathbb{A}'}\Big(\max_{s\in S'_{a'}}\varphi^a_s\Big)B_{a'} + \sum_{i=1}^{m}\varphi^a_{(i,1)}b_i + \sum_{\{j|j\rightsquigarrow j_1\}}\rho^a_j T_j.
\end{aligned}$$

As a consequence, with M_{AG} and N_{AG} playing the role of M and N above, we have $(M_{\mathrm{AG}})_{a,a'} = \max_{s\in S_{a'}}\varphi^a_s$ and $(N_{\mathrm{AG}})_a = \sum_{i=1}^{m}\varphi^a_{(i,1)}b_i + \sum_{\{j|j\rightsquigarrow j_1\}}\rho^a_j T_j$.

Example 5. *We compute backlog bounds for arcs $a_1=(4,2)$ and $a_2=(2,1)$ and obtain.*

$$\begin{cases} B_{a_1} & \le N^{a_1}_{\mathrm{AG}} + (\varphi^{a_1}_{1,2}\vee\varphi^{a_1}_{2,2})B_{a_1} + \varphi_{3,2}B_{a_2}\\ B_{a_2} & \le N^{a_2}_{\mathrm{AG}} + \varphi^{a_2}_{1,2}B_{a_1}. \end{cases}$$

A sufficient condition for the stability is then given by $(\varphi^{a_1}_{1,2}\vee\varphi^{a_1}_{2,2}) + \varphi_{3,2}\varphi^{a_2}_{1,2} < 1$.

It is not possible to compare the stability bound with M^{AG} with M^{TD} or M^{SD}: there are examples where the stability bound will be better, for the unidirectional ring for example, and some examples where it will be worse, like for the bidirectional ring. In the next section, we present those two examples that illustrate the advantages and limits of this latter approach.

5.2 Examples

5.2.1 Stability of the unidirectional ring

Consider a ring with n nodes. Its induced graph is $\mathcal{G}$ with $\mathbb{A}=\{(i,i+1), i\le n-1\}\cup\{(n,1)\}$. The transformation into a tree gives a tandem networks by removing arc $(n,1)$. Flows are decomposed in either one flow or two flows. Grouping flows that cross this arc enables to show the stability of the unidirectional ring.

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

Proof. We consider matrix M_{AG} and take $\mathbb{A}^{\mathrm{r}}=\{(n,1)\}$. With I the set of flows that circulate through arc $(n,1)$, $S_{(n,1)}=\{(i,2)\ |i\in I\}$. When computing the worst-case backlog at arc a, the flows of interests are flows $f_{(i,1)}$ for $i\in I$ and

$$B_a \le \max_{s\in S_a}\varphi^a_s B_a + C,$$

where C is a constant there is no need to explicit in this proof. So it remains to show that for all $s\in S_a$, $\varphi^a_s<1$. As $(i,2)\in S_a$ is not a flow of interest, $\varphi_{(i,2)}=\xi_1^{\pi_i(\ell_i)}$. Observe from Algorithm 1

how ξ_j^ℓ are computed: because of the local stability, $R_n > r_n^n + r_n^*$, so $\xi_n^n < 1$. Now assume that $\xi_{j\bullet}^k < 1$ (lines 7-11). Either $\xi_j^k = \xi_{j\bullet}^k < 1$, or $\xi_j^\ell = (r_j^* + \sum_{\ell' \in k^\bullet \rightsquigarrow n} \xi_{j\bullet}^{\ell'} r_j^{\ell'})/(R_j - \sum_{\ell \in j \rightsquigarrow k} r_j^{\ell'}) \leq (r_j^* + \sum_{\ell' \in k^\bullet \rightsquigarrow n} r_j^{\ell'})/(R_j - \sum_{\ell \in j \rightsquigarrow k} r_j^{\ell'}) < 1$, as from local stability condition. As a consequence, for all j and ℓ, $\xi_j^\ell < 1$ and $\max_{s \in S_a} \varphi_s^a < 1$, and $B_a \leq C(1 - \max_{s \in S_a} \varphi_s^a)^{-1}$, ensuring the stability of the network. □

This result has already been proved under stronger assumptions: in [23] when servers are constant-rate servers and in [17] when 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.

5.2.2 The bi-directional ring

An example where grouping according to the arcs is not be efficient is the bi-directional ring with n servers. Suppose the network is crossed by $2n$ flows of length n: $\pi_1 = \langle 1, 2, \ldots, n \rangle$, $\pi_{n+1} = \langle n, n-1, \ldots, 1 \rangle$, $\pi_i = \langle i, i+1, \ldots, n, 1 \ldots, i-1 \rangle$ and $\pi_{n+i} = \langle i, i-1, \ldots, 1, n \ldots, i+1 \rangle$ for $i = 2, \ldots, n$. The tree decomposition is obtained by keeping arcs $\{(i, i+1), i \leq n-1\}$ and the path obtained after the decomposition are the one obtained for the unidirectional ring for $f_1, \ldots, f_n$, and flows of length 1 for the other paths.

With this decomposition, we can never ensure stability: let us look at the coefficient $\varphi_{a'}^a$ that are computed. Consider arc $a = (2, 1)$ for example. Among the flows of interest are the flows $f_{(i,k)}^a$ of path $\langle 2 \rangle$, with $k \neq 1$, so $\varphi_{(i,k)}^a = 1$. This means that $(M_{\mathrm{AG}})_{(2,1),(3,2)} = 1$, and the similarly, $(M_{\mathrm{AG}})_{(j,j-1),(j+1,j)} = 1$ and $(M_{\mathrm{AG}})_{(1,n),(2,1)} = 1$. There is a cycle of coefficients 1 in the matrix: the spectral radius of M_{AG} is at least 1.

More generally, grouping according to the arcs will never ensure the stability if in matrix M_{AG} it is possible to find a cycle with weights one on all its arcs. As a consequence, intermediate solution between no grouping of flows and grouping among the arcs might lead to better solutions. For example, in the case of the bi-directional ring, a better solution would be to group flows for the removed arc $(n, 1)$ only, and not group the other flows.

5.3 Two-stage optimization problem

We have seen in through the examples of the previous paragraph that different stability conditions and performance bounds can be found, depending on how the network is decomposed. Following the approach of Equation (5), it is possible to combine the optimization problems:

$$\begin{array}{ll} \text{Maximize} & Q\mathbf{b}' + C \\ \text{such that} & \mathbf{b}' \leq \mathbf{b}, \sum_{s \in S_a} b'_s \leq B_a \\ & \mathbf{b} \leq M_{\mathrm{TD}}\mathbf{b} + N_{\mathrm{TD}},\ \mathbf{B} \leq M_{\mathrm{AG}}\mathbf{B} + N_{\mathrm{AG}}. \end{array}$$

This formulation slightly differs from the one in Equation (5): constraint '$b'_s \leq B_a \quad \forall s \in S_a$" has been replaced by "$\sum_{s \in S_a} \mathbf{b}'_s \leq B_a$". Indeed, by a reasoning similar to that of Paragraph 5.1.3, we can fix $a = (i, j) \in \mathbb{A}^r$. If the worst-case backlog at server i for flows crossing a is B_a this means that when this worst-case happens, there is no data of these flows in the rest of the network (which would deny the maximality of B_a). Consider a flow f_s, $s \in S_a$. Its amount of data in arc a is x_s, and its arrival rate r_s. Data cannot arrive faster than r_s, so from the time of worst-case backlog, flow f_s is γ_{x_s, r_s}-constrained.

5.4 General arrival and service curves

Beyond the linear model, it should be possible to obtain tighter bounds by using more general arrival and service curves. For example, stair-case functions, or piece-wise linear arrival curves and service curves.

A first remark is that the stability conditions given here only depend on the arrival and service rates, then, in the case it is possible to refine these results to more general curves (as it is for the SD method), no better stability condition can be inferred. Indeed, a general curve can usually be lower and upper-bounded by two token-bucket curves, inducing a lower and an upper-bound of the network by two linear models with the same stability condition.

The second remark is that it would still be possible to improve the performance bounds. To our knowledge, there is no evidence in the literature that the equation $\boldsymbol{\alpha} = H(\boldsymbol{\alpha})$ has a unique fix-point in the general case. One safe solution is to compute the greatest fix-point by iterations methods. The first step of this approach is to find an upper bound of that greatest fix-point. This can be done by computing the fix point of that equation in the linear model (by bounding the arrival and service curves by linear curves), and the second step is to iterate from that point for refining the performance bounds. At each iteration, the performance bound obtained is an upper bound of the performance of the network.

6 Comparison and numerical experiments

The different approaches have been implemented in Python and run on a basic laptop. We will not comment on the computational time as all those algorithms are polynomial, and the number of constraints of our linear programs are linear in the size of the networks.

We call SD the server decomposition method, TD the tree decomposition method, AG the arc grouping method and 2S 2-stage method.

We compare those methods on three examples: the unidirectional ring, the bidirectional ring and a 3-ring network.

In the experiments we assume that the utilization rate of the network is $U = \min_{j=1}^{n} U_j$. flows have uniform parameters: $b_i = 1kb$, $r_i = 1kb.s^{-1}$ for all $i \in \mathbb{N}_m$, and $T_j = 10ms$ for all $j \in \mathbb{N}_n$. Only the service rate will vary in function of an utilization rate: the utilization rate of server j is $U_j = \frac{\sum_{i \in \mathrm{Fl}(j)} r_i}{R_j}$, and

6.1 The unidirectional ring

We now consider the example of the unidirectional ring described in Section 5.2.1 with $n = 10$.

Figure 4 shows the backlog guarantee at server n of flow 1 for uniform traffic: servers all have the same service rate $R = 10/Ukbs^{-1}$. The stability conditions are $U < 0.18$ for SD and $U < 0.62$ for SD, so our methods greatly improves the stability region. We notice that AG is better than TD, so AG and 2S compute the same bounds.

Fig. 5 (left) shows the ratio between the stability bounds with SD and with TD as the number of servers increases on the ring. The ratio grows linearly.

Fig 6 shows the backlog guarantee of flow 1 at server n when the servers have different service rates: every service rate is $R = 20/Ukbs^{-1}$, except $R_9 = R_{10} = 10/Ukbs^{-1}$.

In this case, TD is better than AG for $U < 0.68$, and around the stability limit given by TD, we observe 2S increases faster as the backlog computed with TD grows to infinity. Then 2S and AG compute of course the same bound.

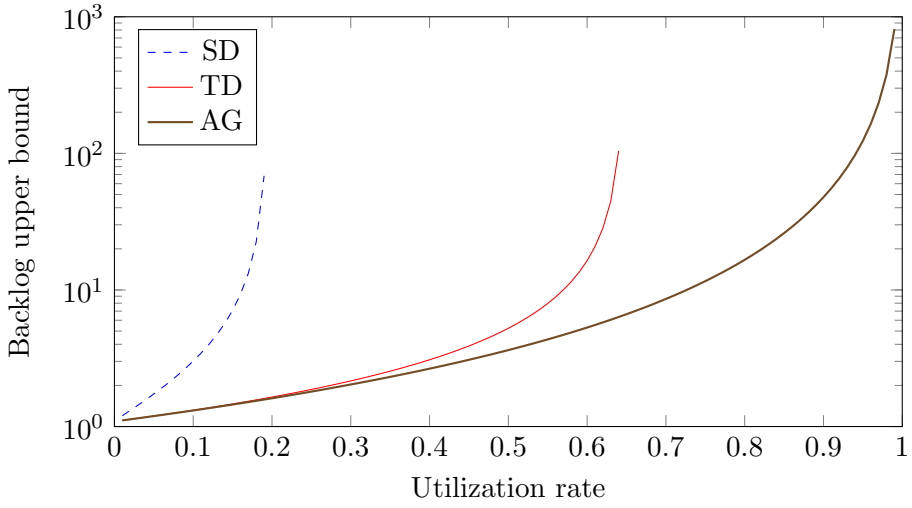

Figure 4: Backlog bound for the unidirectional ring and uniform servers.

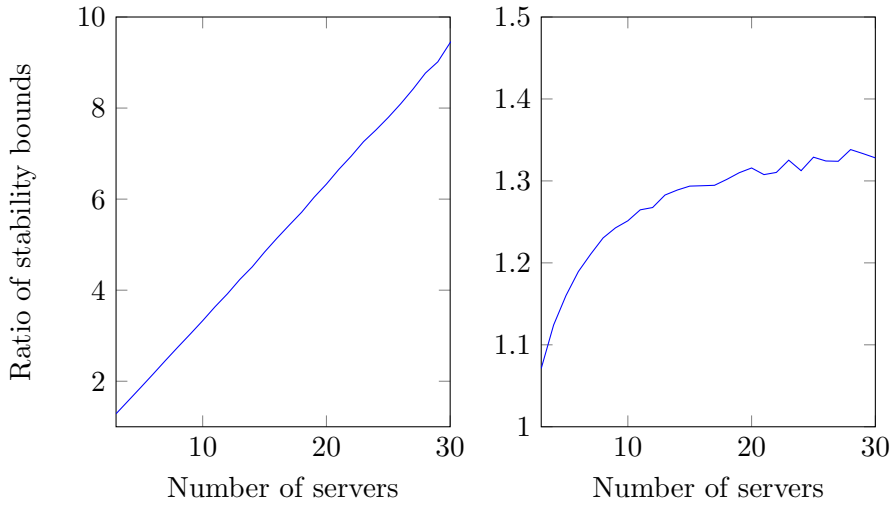

Figure 5: Ratio of stability bounds on the ring with the number of servers grows. Left: unidirectional ring; right: bidirectional ring.

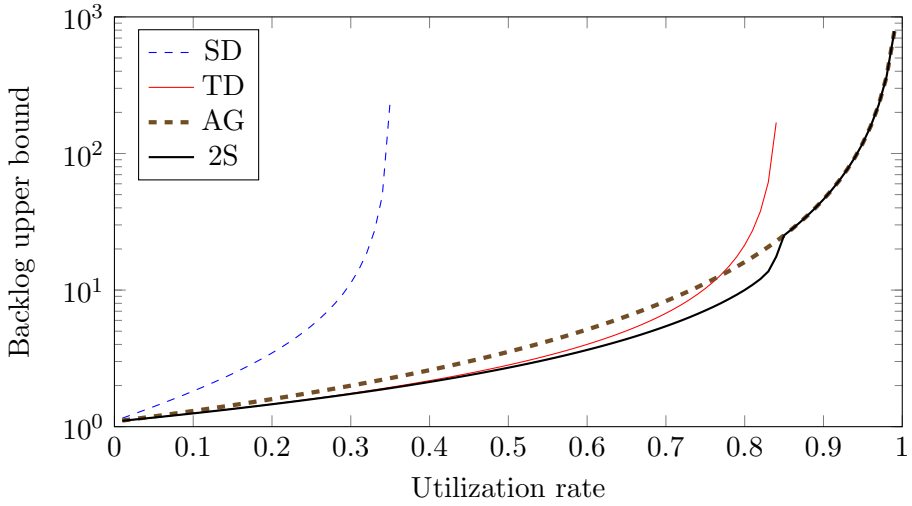

Figure 6: Backlog bound for the unidirectional ring with heterogeneous servers.

6.2 The bidirectional ring

We now consider the example of the bidirectional ring with $n = 10$ as described in Paragraph 5.2.2.

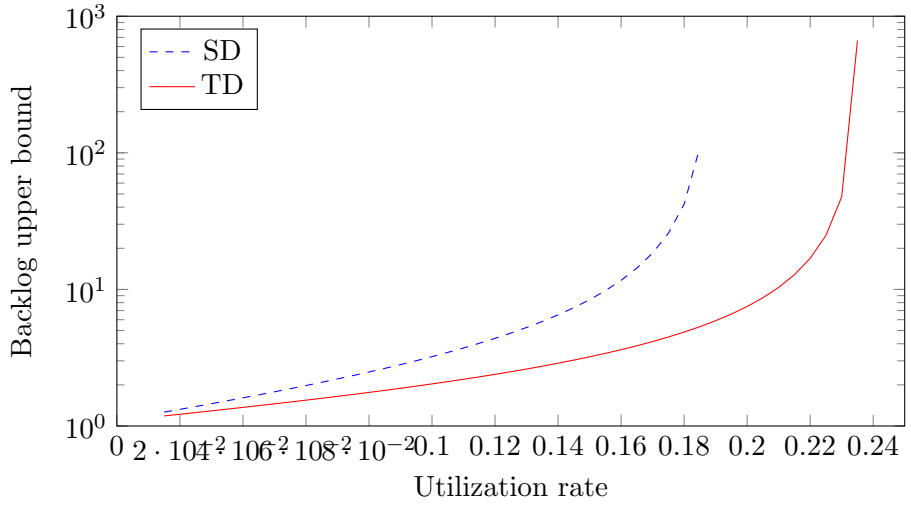

Figure 7: Backlog bounds for the bidirectional ring.

Figure 7 shows the worst-case backlog bound of flow 1 at server n computed by the elementary decomposition and tree transformation. As expected, the stability condition with TD ($U < 0.24$) is improved from SD ($U < 0.19$). The improvement is approximately 25%. Fig. 5 (right) shows the improvement ratio when the number of servers grows. In this case, the improvement seems logarithmic, and for $n = 30$, it is approximately 33%. TD method suffers from having half the flows decomposed in flows of length 1.

6.3 A three-ring example

The bidirectional cycle is not realistic, as in many network are full-duplexed, but there might be several cycles in network. Fig. 8 shows an example of a network composed of three cycles,

and flows circulate along one of the three cycles. Three servers (those depicted) are common to two cycles.

Fig. 9 shows the backlog of a flow when each cycle is made of 10 servers, and flows have length 10, except for one cycle, where we use shorter flows to avoid the problem presented in Paragraph 5.2.2. We can observe that the stability region more than doubles from the SD ($U < 0.34$) to the TD method ($U < 0.73$). The improvement using grouping is smaller ($U < 0.77$), but still sensible.

7 Conclusion

In this article, the recent results from feed-forwards networks can be adapted to improve the performance guarantees and stability conditions of networks with general topology.

Many problems remain open and directions to investigate: finding the transformation of the network that would lead to better guarantees, adapt recent results for example [7] that can be applied for general arrival and service curves. Future works will also include the adaptation to service policies, like the FIFO of static priority policies.

More generally, the stability problem remains open.

References

[1] Carme Àlvarez, Maria J. Blesa, and Maria J. Serna. A characterization of universal stability in the adversarial queuing model. *SIAM J. Comput.*, 34(1):41–66, 2004.

[2] A. Amari, Ahlem Mifdaoui, Fabrice Frances, Jérome Lacan, David Rambaud, and Loic Urbain. AeroRing: Avionics Full Duplex Ethernet Ring with High Availability and QoS Management. In *ERTS*, 2016.

[3] M. Andrews. Instability of fifo in session-oriented networks. In *Proceedings of SODA'00*, 2000.

[4] M. Andrews. Instability of FIFO in the permanent sessions model at arbitrarily small network loads. In *Proceedings of SODA'07*, 2007.

[5] Matthew Andrews, Baruch Awerbuch, Antonio Fernández, Frank Thomson Leighton, Zhiyong Liu, and Jon M. Kleinberg. Universal-stability results and performance bounds for greedy contention-resolution protocols. *J. ACM*, 48(1):39–69, 2001.

[6] Steffen Bondorf and Jens B. Schmitt. Boosting sensor network calculus by thoroughly bounding cross-traffic. In *Proceedings of INFOCOM 2015*, 2015.

[7] Steffen Bondorf and Jens B. Schmitt. Improving cross-traffic bounds in feed-forward networks - there is a job for everyone. In *MMB & DFT*, pages 9–24, 2016.

[8] Allan Borodin, Jon M. Kleinberg, Prabhakar Raghavan, Madhu Sudan, and David P. Williamson. Adversarial queuing theory. *J. ACM*, 48(1):13–38, 2001.

[9] A. Bouillard, L. Jouhet, and É. Thierry. Comparison of different classes of service curves in network calculus. In *WODES*, pages 316–321, 2010.

[10] A. Bouillard, L. Jouhet, and É. Thierry. Tight Performance Bounds in the Worst Case Analysis of Feed Forward Networks. In *INFOCOM'10*, 2010.

0.01 1.04843018865 1.03003970439 1.03157491266 1.03003956473
0.015 1.05803963769 1.03516270593 1.03748103313
1.03516241589
0.02 1.06792652515 1.04035641182 1.04346858223 1.04035624789
0.025 1.07810254997 1.04562232682 1.04953926172
1.04562207545
0.030000000000000002 1.08858010879 1.05096199835 1.05569487794
1.05096160991
0.035 1.09937235137 1.05637701805 1.06193739419
1.05637638807
0.04 1.11049324151 1.06186902327 1.06826866547 1.06186813863
0.045 1.12195762415 1.06743969883 1.07469068152
1.0674387523
0.049999999999999996 1.13378129941 1.07309077863 1.08120549712
1.07308990444
0.05499999999999999 1.14598110437 1.07882404752 1.08781521923
1.07882272777
0.05999999999999999 1.15857500368 1.08464134315 1.09452201991
1.08463969874
0.06499999999999999 1.17158218999 1.09054455789 1.10132813883
1.09054244568
0.06999999999999999 1.18502319562 1.09653564095 1.10823584836
1.09653285715
0.075 1.19892001688 1.10261660044 1.1152476146
1.10261317395
0.08 1.21329625282 1.10878950563 1.12236579679 1.10878542261
0.085 1.22817726043 1.11505648932 1.1295929672
1.11505155562
0.09000000000000001 1.24359032848 1.12141975021 1.13693173532
1.12141367639
0.09500000000000001 1.25956487298 1.12788155553 1.14438479765
1.12787435193
0.10000000000000002 1.27613265718 1.13444424366 1.15195493842
1.13443550677
0.10500000000000002 1.29332804 1.14111022696 1.15964503295
1.14109974331
0.11000000000000003 1.31118825722 1.14788199468 1.16745805138
1.14786968398
0.11500000000000003 1.32975374069 1.15476211608 1.17539705992
1.15474802568
0.12000000000000004 1.34906848162 1.16175324359 1.18346523043
1.16173672284
0.12500000000000003 1.36918044535 1.16885811626 1.19166584179
1.16883885349
0.13000000000000003 1.39014204638 1.17607956324 1.20000228337
1.17605722353
0.13500000000000004 1.41201069407 1.18342050755 1.20847806057
1.18339469766
0.14000000000000004 1.43484942177 1.19088396997 1.21709679967
1.19085420287
0.14500000000000005 1.45872761466 1.19847307315 1.22586225209
1.19843930476
0.15000000000000005 1.48372185491 1.20619104591 1.23477830239
1.20615250589
0.15500000000000005 1.50991690695 1.21404122779 1.24384897182
1.21399734712
0.16000000000000006 1.53740687086 1.22202707381 1.25307842494
1.22197728291

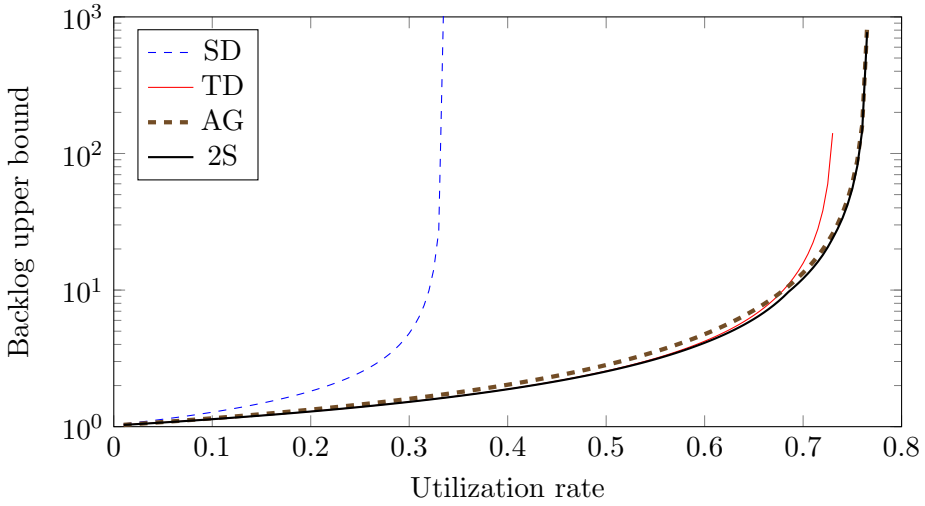

Figure 9: Backlog bound for the three-ring example.

[11] Anne Bouillard and Thomas Nowak. Fast symbolic computation of the worst-case delay in tandem networks and applications. *Perform. Eval.*, 91:270–285, 2015.

[12] M. Boyer, N. Navet, X. Olive, and É. Thierry. The PEGASE project: precise and scalable temporal analysis for aerospace communication systems with network calculus. In *ISOLA'10*, 2010.

[13] C.-S. Chang. *Performance Guarantees in Communication Networks.* TNCS, Springer-Verlag, 2000.

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

[15] R.L. Cruz. Quality of service guarantees in virtual circuit switched networks. *IEEE Journal on selected areas in communication*, 13:1048–1056, 1995.

[16] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[17] J.-Y. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, volume LNCS 2050. Springer-Verlag, 2001. revised version 4, May 10, 2004.

[18] Zvi Lotker, Boaz Patt-Shamir, and Adi Rosén. New stability results for adversarial queuing. *SIAM J. Comput.*, 33(2):286–303, 2004.

[19] J. M. McManus and K. W. Ross. Video-on-demand over ATM: Constant-rate transmission and transport. *IEEE J.Sel. A. Commun.*, 14(6):1087–1098, September 2006.

[20] Francesco De Pellegrini, David Starobinski, Mark G. Karpovsky, and Lev B. Levitin. Scalable cycle-breaking algorithms for gigabit ethernet backbones. In *Proceedings IEEE INFOCOM*, 2004.

[21] G. Rizzo and J.-Y. Le Boudec. Stability and delay bounds in heterogeneous networks of aggregate schedulers. In *Proceedings of INFOCOM'2008*, 2008.

[22] David Starobinski, Mark G. Karpovsky, and Lev Zakrevski. Application of network calculus to general topologies using turn-prohibition. In *Proceedings IEEE INFOCOM*, 2002.

[23] Leandros Tassiulas and Leonidas Georgiadis. Any work-conserving policy stabilizes the ring with spatial re-use. *IEEE/ACM Trans. Netw.*, 4(2):205–208, 1996.

A Proof of Theorem 4

To avoid introducing too many notations, we first prove the result for tandem networks ($\mathbb{A} = \{(i, i+1) \mid i \in \mathbb{N}_{n-1}\}$). We then explain how to adapt it to sink-trees.

We prove the theorem by a backward induction on the servers. Let us denote by $B_j(x_j, \ldots, x_n, x^*)$ the backlog at time t_{n+1} in server n when the backlog transmitted at time t_j by server $j-1$ to server j is x_k for the flows served by server $j-1$ and that end at server k ($k \geq j$) and x^* for the flows of interest crossing server $j-1$.

We will use the additional notations

- $b_j^k = \sum_{\{i \notin I,\ \pi_i = j \rightsquigarrow k\}} b_i$ is the size of the burst arriving at server j at time t_j belonging to flows starting at server j and ending at server k;
- $b_j^* = \sum_{\{i \in I \mid\ \pi_i(1) = j\}} b_i$ is the burst of the flows of interests starting at server j.

If we are able to compute B_j for all j, the worst-case backlog is $B = B_1(0, \ldots, 0)$. We will show by induction that:

(**A**) B_j is linear in the x_k, T_k, b_ℓ^k, b_ℓ^*, $k \geq \ell \geq j$ and in x^*. More precisely we can write

$$B_j((x_k)_{k \geq j}, x^*) = C_j + x^* + \sum_{k=j}^{n} \xi_j^k x_k,$$

where ξ_j^k only depends on the R_k's and $r_i^{(*)}$'s, and C_j is a polynomial of degree 1 in T_k, b_ℓ^k, b_ℓ^*, $k \geq \ell \geq j$ (with coefficients depending on the R_k's and $r_i^{(*)}$'s only).

(**B**) $\xi_j^k \leq \xi_j^{k+1}$.

This inequality (**B**) is quite intuitive: the coefficient ξ_j^k roughly corresponds to quantity of data produced by a flow starting at j and ending at k the rate grows when the length of the path grows, as there is more chance to meet a slower server.

A.1 Initialization - computation of B_n

B_n only depends on the burst that is transmitted at time t_{n+1}, which we note x_n^n for the flows that are not of interest and x^* for the flows of interest. The worst-case backlog for the flows of interests is obtained when all data from the other flows have been served and none of the flows of interests:

$$\begin{aligned} B_n(x_n^n, x^*) =& b_n^* + x^* + r_n^*\left(T_n + \frac{x_n^n + b_n^n + r_n^n T_n}{R_n - r_n^n}\right) \\ =& b_n^* + x^* + r_n^* T_n + \xi_n^n Q_n^n \\ =& C_n + x^* + \xi_n^n Q_n^n, \end{aligned}$$

with $Q_n^n = x_n^n + b_n^n + r_n^n T_n$, $\xi_n^n = \frac{r_n^*}{R_n - r_n^n}$ and $C_n = b_n^* + r_n^* T_n$.

A.2 Inductive step - computation of B_j from B_{j+1}

Suppose that $B_{j+1}(x_{j+1},\dots,x_n,x^*) = C_{j+1} + x^* + \sum_{k=j}^n \xi_j^k x_k$.

$B_j^k(x_j^j,\dots,x_j^n,x^*)$ is computed the following way: it takes time δ to serve flows ending at servers $j,\dots,k$, and data from any other flow is instantaneously transmitted to server $j+1$. This quantity of data is $x_{j+1}^\ell = b_j^\ell + x_j^\ell + r_j^\ell\delta$ for $\ell > k$ where δ satisfies

$$\sum_{\ell=j}^{k}(x_j^\ell + b_j^\ell + r_j^\ell\delta) = R_j(\delta - T_j)_+,$$

i.e., $\delta = T_j + \sum_{\ell=j}^k Q_j^\ell/\left(R_j - \sum_{\ell=j}^k r_j^\ell\right)$ with $Q_j^k = b_j^k + x_j^k + r_j^k T_j$. For $\ell > k$, the amount of data transmitted to server $j+1$ by flows (not of interest) ending at server $k \le j$ is

$$x_{j+1}^\ell = Q_j^\ell + r_j^\ell \frac{\sum_{\ell=j}^k Q_j^\ell}{R_j - \sum_{\ell=j}^k r_j^\ell}.$$

Then the backlog at server n can be expressed from B_{j+1}:

$$\begin{aligned}
B_j^k(x_j^j,\dots,x_j^n,x^*) =& B_{j+1}(0\dots,0,x_{j+1}^{k+1},\dots,x_{j+1}^n, b_j^* + x^* + r_j^*\delta).\\
=& C_{j+1} + b_j^* + x^* + r_j^*\left(T_j + \frac{\sum_{\ell=j}^k Q_j^\ell}{R_j - \sum_{\ell=j}^k r_j^\ell}\right) + \sum_{\ell>k}\xi_{j+1}^\ell\left(Q_j^\ell + r_j^\ell\frac{\sum_{\ell=j}^k Q_j^\ell}{R_j - \sum_{\ell=j}^k r_j^\ell}\right)\\
=& C_j + x^* + r_j^*\frac{\sum_{\ell=j}^k Q_j^\ell}{R_j - \sum_{\ell=j}^k r_j^\ell} + \sum_{\ell>k}\xi_{j+1}^\ell\left(Q_j^\ell + r_j^\ell\frac{\sum_{\ell=j}^k Q_j^\ell}{R_j - \sum_{\ell=j}^k r_j^\ell}\right)\\
=& C_j + x^* + \sum_{\ell=j}^k\left(\frac{r_j^* + \sum_{\ell>k}\xi_{j+1}^\ell r_j^\ell}{R_j - \sum_{\ell=j}^k r_j^\ell}\right)Q_j^\ell + \sum_{\ell>k}\xi_{j+1}^\ell Q_j^\ell
\end{aligned}$$

with $C_j = C_{j+1} + b_j^* + r_j^* T_j$.

Note the other scenarios should have been taken into account, when part of the flows ending at server k is served and another part is transmitted to server $j+1$. It can be easily proved that these "mixed" scenarios cannot lead to strictly larger worst-case backlog (see Lemma 4 in [11] for a proof).

Lemma 3. *There exists k such that $B_j = B_j^k$ (that is, for all $x_j,\dots,x_n,x^*$, we have $B_j(x_j,\dots,x_n,x^*) = B_j^k(x_j,\dots,x_n,x^*)$) and for all k, $\xi_j^k \ge \xi_j^{k+1}$.*

Proof. The proof is also by induction. We prove that $\forall j$, we have the equivalence

$$\begin{aligned}
B_j = B_j^k \Leftrightarrow \forall k' > k\ \xi_{j+1}^{k'} &> \frac{\sum_{i\le j} r_i^* + \sum_{\ell>k'}\xi_{j+1}^\ell r_j^\ell}{R_j - \sum_{\ell=j}^{k'} r_j^\ell}\\
\text{and } \xi_{j+1}^k &\le \frac{\sum_{i\le j} r_i^* + \sum_{\ell>k}\xi_{j+1}^\ell r_j^\ell}{R_j - \sum_{\ell=j}^{k} r_j^\ell}.
\end{aligned}$$

Assertion **(B)** is proved at the same time: assuming that **(B)** is satisfied for server $j+1$, we will prove it for server j.

The equivalence above also state that if for all $k' > k$ $B_j \ne B_j^{k'}$, then $B_j^n \le B_j^{n-1} \le \dots \le B_j^k$.

Indeed, we have the equivalence $B_j \ne B_j^n \Leftrightarrow \xi_{j+1}^n > \frac{r_j^*}{R_j - \sum_{\ell=j}^n r_j^\ell} \Leftrightarrow \frac{r_j^*}{R_j - \sum_{\ell=j}^n r_j^\ell} < \frac{r_j^* + \xi_{j+1}^n r_j^n}{R_j - \sum_{\ell=j}^{n-1} r_j^\ell}$

But

$$B_j^n = C_j + x^* + \sum_{\ell \le n} \frac{\sum_{i \le j} r_i^*}{R_j - \sum_{\ell=j}^n r_j^\ell} Q_j^\ell$$

and

$$B_j^{n-1} = C_j + x^* + \sum_{\ell \le n-1} \frac{\sum_{i \le j} r_i^* + \xi_{j+1}^n r_j^\ell}{R_j - \sum_{\ell=j}^{n-1} r_j^\ell} Q_j^\ell + \xi_{j+1}^n Q_j^n.$$

So $B_j^n \le B_j^{n-1}$. The next steps can be shown similarly.

Let us assume that $B_j^n \le B_j^{n-1} \le \cdots \le B_j^k$.

The coefficient of Q_j^ℓ in $B_j^{k'}$ for all $k' \le k$ and $\ell > k$ is ξ_{j+1}^ℓ and that of Q_j^k is

- either $\frac{r_j^* + \sum_{\ell > k} \xi_{j+1}^\ell r_j^\ell}{R_j - \sum_{\ell=j}^k r_j^\ell}$ (for B_j^k)
- or ξ_{j+1}^k (for B_j^ℓ, $\ell < k$).

Suppose that

$$\frac{r_j^* + \sum_{\ell > k} \xi_{j+1}^\ell r_j^\ell}{R_j - \sum_{\ell=j}^k r_j^\ell} \ge \xi_{j+1}^k. \tag{10}$$

For $\ell \le k' < k$, the coefficient of Q_j^ℓ in $B_j^{k'}$ is

$$\begin{aligned}
\frac{r_j^* + \sum_{\ell > k'} \xi_{j+1}^\ell r_j^\ell}{R_j - \sum_{\ell=j}^{k'} r_j^\ell} &\le \frac{r_j^* + \sum_{\ell'=k+1}^n r_j^{\ell'} \xi_{j+1}^{\ell'} + \sum_{\ell'=k'+1}^k r_j^{\ell'} \xi_{j+1}^k}{R_j - \sum_{\ell'=j}^{k'} r_j^{\ell'}} \\
&\le \frac{r_j^* + \sum_{\ell'=k+1}^n r_j^{\ell'} \xi_{j+1}^{\ell'} + \sum_{\ell'=k'+1}^k r_j^{\ell'} \frac{r_j^* + \sum_{\ell > k} \xi_{j+1}^\ell r_j^\ell}{R_j - \sum_{\ell=j}^k r_j^\ell}}{R_j - \sum_{\ell'=j}^{k'} r_j^{\ell'}} \\
&= \frac{r_j^* + \sum_{\ell'=k+1}^n r_j^{\ell'} \xi_{j+1}^{\ell'}}{R_j - \sum_{\ell'=j}^{k'} r_j^{\ell'}} \left(1 + \frac{\sum_{\ell'=k'+1}^k r_j^{\ell'}}{R_j - \sum_{\ell=j}^k r_j^\ell}\right) \\
&= \frac{r_j^* + \sum_{\ell'=k+1}^n r_j^{\ell'} \xi_{j+1}^{\ell'}}{R_j - \sum_{\ell=j}^k r_j^\ell}
\end{aligned}$$

which is the coefficient of Q_j^ℓ in B_j^k. The first inequality uses (**B**) for server $j+1$ and the second uses Inequality (10).

Moreover, for $k' < \ell < k$, the coefficient of Q_j^ℓ in $B_j^{k'}$ is

$$\xi_{j+1}^{k'} \le \xi_{j+1}^k \le \frac{r_j^* + \sum_{\ell > k} \xi_{j+1}^\ell r_j^\ell}{R_j - \sum_{\ell=j}^k r_j^\ell},$$

so $B_j^k \ge B_j^{k'}$ and $B_j = B_j^k$.

The same kind of computations lead to $B_j^{k-1} > B_j^k$ if $\xi_{j+1}^k > \frac{r_j^* + \sum_{\ell > k} \xi_{j+1}^\ell r_j^\ell}{R_j - \sum_{\ell=j}^k r_j^\ell}$.

Set ξ_j^k according to the B_j that is computed (that is B_j^k for some k). Then we still have $\xi_j^\ell \ge \xi_j^{\ell+1}$ and (**B**) is true for server j. □

Finally, we have

$$B = \sum_{j \le n} (r_j^* T_j + \sum_{k \ge j} \xi_j^k r_j^k T_j) + \sum_{i \le n} b_i^* + \sum_{j \le k \le n} \xi_j^k b_j^k.$$

A.3 Adaptation to trees

Consider now a sink-tree, the above analysis is still valid, and each branch of the tree can be analysed independently: if a server has several predecessors, then the optimization will be for each of the on disjoint sets of servers and flows.

The algorithm still runs in polynomial time.